Jetter automation




# User Manual

## JC-360 - Controller

60880622

We automate your success.

Item # 60880622
Revision 1.26.1
November 2015 / Printed in Germany

This document has been compiled by Jetter AG with due diligence, and based on the known state of the art.

In the case of modifications, further developments or enhancements to products shipped in the past, a revised document will be supplied only if required by law, or deemed appropriate by Jetter AG. Jetter AG shall not be liable for errors in form or content, or for missing updates, as well as for damages or disadvantages resulting from such failure.

The logos, brand names, and product names mentioned in this document are trademarks or registered trademarks of Jetter AG, of associated companies or other title owners and must not be used without consent of the respective title owner.

**Address**

How to contact us:

Jetter AG
Graeterstrasse 2
71642 Ludwigsburg
Germany

| | |
|---|---|
| Phone - Switchboard: | +49 7141 2550-0 |
| Phone - Sales: | +49 7141 2550-433 |
| Phone - Technical Hotline: | +49 7141 2550-444 |
| Fax - Sales: | +49 7141 2550-484 |
| E-mail - Sales: | sales@jetter.de |
| E-mail - Technical Hotline: | hotline@jetter.de |


**Assignment to product**

This User Manual is an integral part of JC-360:

Type: _____

Serial #: _____

Year of manufacture: _____

Order #: _____

CE

To be entered by the customer:

Inventory #: _____

Place of operation: _____

**Significance of this User Manual**

This document is an integral part of the JC-360:

- Keep this document in a way that it is always at hand until the JC-360 will be disposed of.
- Pass this document on if the JC-360 is sold or loaned/leased out.

In any case you encounter difficulties to clearly understand the contents of this document, please contact Jetter AG.

We would appreciate any suggestions and contributions on your part and would ask you to contact us at the following e-mail address: info@jetter.de. Your feedback will help us produce manuals that are more user-friendly, as well as address your wishes and requirements.

This document contains important information on the following topics:

- Transport
- Mounting
- Installation
- Programming
- Operation
- Maintenance
- Repair

Therefore, you must carefully read, understand and observe this document, and especially the safety instructions.

In the case of missing or inadequate knowledge of this document Jetter AG shall be exempted from any liability. Therefore, the operating company is recommended to obtain the persons' confirmation that they have read and understood this manual in writing.

# Table of Contents


**1 Safety instructions 13**

Basic safety instructions ....14
Instructions on EMI ....16

**2 Product description and design 19**

Product description of the JC-360....20
Parts and interfaces of the controller JC-360 ....21
Order reference/options ....22
List of documentation ....23
Accessories for the JX3 system....25
Physical dimensions ....26

**3 Identifying 27**

**3.1 Identification by means of the nameplate ....28**
JC-3xx: Nameplate ....29
**3.2 Electronic Data Sheet EDS....30**
EDS file ....31
EDS registers....35
**3.3 Version registers....37**
Hardware revisions ....38
Software versions....39

**4 Mechanical and electrical installation 41**

**4.1 Interfaces ....42**
Power supply terminal X10 ....43
Serial interface port X11....44
Ethernet ports - Female connectors X14, X15....47
**4.2 JX2 system bus interface - Female connector X19 ....48**
Female connector X19 - Pin assignment....49
JX2 system bus cable specification ....51
Line length and baud rate of the JX2 system bus....53
**4.3 LEDs indicating various states....54**
LEDs of the controller ....55
LEDs of the controller during boot process....57
LED indications on the controller after an IP address conflict has been detected....59
Status LEDs - Ethernet interface....60
**4.4 Controls and SD memory card ....61**
Function description of mode selector S11 ....62
SD card slot X61 ....64
**4.5 Installing, replacing and removing the module ....66**
Installing the JC-360 on a DIN rail ....67
Replacing the controller JC-360....68
Removing the JC-360 from the DIN rail....70
**4.6 IP configuration....71**
Factory settings....72
The configuration memory ....73
The configuration file....74

Configuration registers ... 78
Changing the IP address of the controller ... 79
Setting the default IP address 192.168.10.15 ... 80
Setting the IP address via configuration file ... 81
Setting the IP address via configuration file and DIP switch ... 82
Setting the IP address via registers to be non-volatile ... 84
Setting the IP address during runtime ... 86
IP address in the GNN operating mode ... 87
Using names for IP addresses ... 89
**4.7 Engineering of JX3 station equipped with a JC-360 ... 91**
**4.7.1 Limitations to be taken into account when engineering a JX3 station ... 92**
Limitations of the maximum number of modules ... 95
Limitations of the modules' data exchange rates ... 96
Limitation depending on the power consumption of the modules ... 99
**4.8 Configuring the JX2 system bus ... 104**
**4.8.1 Wiring the JX2 system bus ... 105**
Line length and baud rate of the JX2 system bus ... 106
JX2 system bus topology ... 107
Power supply of JX2-I/O modules ... 108
Power supply of JX2 slave modules ... 110
**4.8.2 Third-party CANopen® modules ... 111**
Product description - Module BWU1821 by Bihl+Wiedemann ... 112
Product description - ECOSTEP® ... 113
Product description - Festo CPV-Direct ... 114
Product description - Festo CPX terminals ... 115
Product description - Festo CPX-CP interface ... 117
Product description - Festo CPX-CMAX--1 ... 118
Product description - Festo CPX-CMPX ... 119
Product description - Festo MTR-DCI ... 120
Product description - Festo SFC-DC ... 121
Product description - Festo SFC-LAC ... 122
Product description - Festo SFC-LACI ... 123
Product description - Lenze 8200 vector ... 124
Product description – maxon EPOS ... 125
Product description - Milan drive ... 126
Product description – SMC EX120 ... 127
Product description – SMC EX250 ... 128
Product description - WAGO I/O-System 750 ... 129
**4.9 Connecting displays and HMIs ... 130**
Overview of displays and HMIs ... 131
Connecting a display or HMI ... 132
Connecting several displays or HMIs: Multi-display mode ... 133
Multi-display mode - Wiring ... 134
Interface cable JC-DK-Xm ... 136
Interface cable KAY_0386-xxxx ... 138
Interface cable KAY_0533-0025 ... 140

**5 Initial commissioning 143**

Preparatory work for initial commissioning ... 144
Initial commissioning of a JC-360 ... 145
Configuring error states ... 147
Configuration in JetSym ... 148

**6 File system 153**

**6.1 Properties ..... 154**
Flash disk - Properties ..... 155
SD card - Properties ..... 156
**6.2 User administration ..... 157**
Administration of users ..... 159
As-delivered condition/Predefined users and keys ..... 161
Assigning locks ..... 162
Assigning names to keys/locks ..... 164
**6.3 Reviewing the flash disk capacity used ..... 166**
Flash disk capacity used ..... 167
**6.4 Operating system update and application program ..... 170**
**6.5 Formatting and checking ..... 171**
Formatting the flash disk ..... 172
Formatting the SD card ..... 173
Checking the SD card ..... 174

**7 FTP server 175**

Logon ..... 176
Example: Windows FTP client ..... 177

**8 FTP client 179**

**8.1 Programming ..... 180**
Initializing the FTP client ..... 181
Establishing a connection to the FTP server ..... 182
Terminating a connection ..... 184
Reading a file ..... 185
Writing to a file ..... 187
Deleting a file ..... 189
Changing directories ..... 191
Creating a directory ..... 193
Deleting directories ..... 195
Determining the current directory ..... 197
**8.2 Registers ..... 199**
Register numbers ..... 200
Registers - Description ..... 201

**9 HTTP server 205**

**9.1 Server Side Includes ..... 206**
First entry in the HTML file ..... 207
Inserting real-time controller values ..... 208
Example of an HTML page ..... 213

**10 Programming 215**

Abbreviations, module register properties and formats ..... 216
**10.1 Memories - Overview ..... 217**
Operating system memory ..... 218
File system memory ..... 219
Application program memory ..... 220
Memory for volatile application program variables ..... 221

Memory for non-volatile application program registers ... 222
Memory for non-volatile application program variables ... 223
Registers on I/O modules ... 224
Memory for non-volatile registers on the backplane module ... 225
Special registers ... 226
Inputs and outputs ... 227
Flags ... 228
**10.2 Register and I/O numbers with a JC-3xx ... 229**
Registers and module registers ... 230
Register and I/O numbers of JX3 modules connected to a JC-3xx ... 232
Register numbers of JX2 slave modules connected to the JX2 system bus ... 233
Registers and I/O numbers of JX2-I/O modules on the JX2 system bus ... 234
Register and I/O numbers of IP67-I/O modules on the JX2 system bus ... 235
Registers and I/O numbers of CANopen® modules on the JX2 system bus ... 236
Register and I/O numbers of JX3 modules connected to a JX3-BN-ETH ... 237
Registers and I/O numbers of JX3 nodules from the JX3-BN-ETH perspective ... 239
**10.3 Jetter Ethernet system bus ... 240**
The Global Node Number ... 242
**10.3.1 Acyclic data interchange ... 243**
Command group NetCopy() ... 245
Command group NetBit() ... 247
Network registers ... 248
Registers located on JX3 modules ... 250
Indirect addressing of remote modules ... 252
Addressing with variable destination window ... 254
Register description - Acyclic data interchange ... 256
**10.3.2 Cyclic data interchange ... 259**
Publish/subscribe ... 261
Publish/subscribe - Registers ... 263
Network registers, network inputs and outputs ... 269
**10.3.3 Hardware Manager ... 272**
Hardware Manager ... 273
**10.3.4 Error handling at the Jetter Ethernet system bus ... 274**
Acyclic data interchange - Error logging ... 275
Error message during CRC computing ... 276
Error message on part of a subscription ... 277
Controller evaluates errors reported by a remote network node ... 278
**10.3.5 JetIPScan - Register description ... 279**
Register numbers ... 280
Global status - Register description ... 281
Warnings and errors - Register description ... 284
Configuration - Register description ... 288
**10.3.6 Administrating the connections of the JetIP/TCP and STX debug server ... 290**
Automatic termination of connections ... 291
Register ... 293
**10.3.7 Executing an ARP request ... 294**
Executing an ARP request ... 295
**10.3.8 JetSync blockage ... 296**
Description of system command registers ... 297
Description of the JetSync blockage system commands ... 300
**10.4 General system registers ... 302**
Description of system command registers ... 303
Description of system commands ... 306
**10.5 Startup delay register ... 311**
Setting the startup delay ... 312

**10.6 Real-time clock (RTC) ... 313**
Technical specifications ... 314
Programming ... 315
**10.7 Runtime registers ... 322**
Description of the runtime registers ... 323
**10.8 Monitoring interface activities ... 325**
Operating principle ... 326
Programming ... 328
**10.9 Controlling HMIs with alphanumeric displays ... 330**
**10.9.1 Connectable HMIs ... 331**
Overview of displays and HMIs ... 332
**10.9.2 Registers ... 333**
Register numbers ... 334
Registers - Overview ... 335
**10.9.3 Configuring the screen size ... 337**
Configuring the screen size manually ... 338
**10.9.4 Displaying texts ... 339**
STX Instructions for displaying texts ... 340
Device numbers ... 342
Cursor position ... 344
Clearing the screen ... 346
**10.9.5 Displaying numerical values ... 348**
STX instruction for displaying numerical values ... 349
Device numbers ... 350
Cursor position ... 352
Setting the length of the display field ... 354
Setting the sign option ... 355
Setting the number of decimal places ... 356
Setting the format of numerical values ... 357
**10.9.6 Entering numerical values ... 358**
STX instruction for the input of numerical values ... 360
Device numbers ... 361
Cursor position ... 363
Setting the length of the input field ... 365
Setting the maximum number of decimal places ... 366
Setting the suggested value ... 367
Polling the number of decimal places ... 368
UserInput - Polling the status ... 369
UserInput - Aborting the instruction ... 370
**10.9.7 Querying the keys ... 371**
Assigning keys ... 372
Registers of basic flag numbers ... 376
**10.9.8 Activating/deactivating LEDs ... 378**
Assigning LEDs ... 379
Registers of LED register numbers ... 380
**10.9.9 Monitor functions ... 382**
Overview of displays and HMIs ... 383
Meaning of keys in monitor function ... 384
Displaying and changing variables ... 385
Configuring the monitor function ... 387
**10.10 Controlling printer and serial interfaces ... 389**
**10.10.1 Supported serial interfaces ... 390**
Overview - Interfaces ... 391
**10.10.2 Registers ... 392**
Register numbers ... 393
Registers - Overview ... 394

**10.10.3 Module numbers - Interface modules ... 395**
Configuring module numbers ... 396
**10.10.4 Outputting texts ... 397**
STX instructions for outputting texts ... 398
Device numbers ... 400
**10.10.5 Outputting numerical values ... 401**
STX instruction for outputting numerical values ... 402
Device numbers ... 403
Setting the length of the display field ... 404
Setting the sign option ... 405
Setting the number of decimal places ... 406
Setting the format of numerical values ... 407
**10.11 JX2 system bus ... 408**
Module array and module codes of connected modules ... 409
JX2 system bus - Baud rate ... 412
Dummy modules on the JX2 system bus ... 414
Monitoring intervals on the JX2 system bus ... 415
JX2 system bus - Description of non-volatile registers ... 417
Register description of modules connected to the JX2 system bus ... 421
Register description - Error logging on the JX2 system bus ... 423
Register description - Timeout and interval times for modules on the JX2 system bus ... 426
Register description - Retry counter for JX2 system bus modules ... 429
Register description - Versions of JX2 system bus drivers ... 430
**10.12 JX3 system bus ... 431**
Module array and module codes of connected modules ... 432
Dummy modules on the JX3 system bus ... 434
JX3 system bus - Description of non-volatile registers ... 435
Register description - Modules detected on the JX3 system bus ... 436
Register description - Error logging on the JX3 system bus ... 437
Register description - Timeout intervals on the JX3 system bus ... 439
Register description - Versions of JX3 system bus drivers ... 440
**10.13 E-mail ... 441**
**10.13.1 Configuring the E-mail feature ... 442**
Structure of the configuration file ... 443
Section [SMTP] ... 444
Section [POP3] ... 446
Section [DEFAULT] ... 448
Configuration file - Examples ... 449
**10.13.2 Creating e-mails ... 450**
Name of the e-mail template file ... 451
Structure of the e-mail template file ... 452
Inserting real-time controller values ... 454
**10.13.3 Sending an e-mail ... 459**
**10.13.4 Registers ... 460**
Overview of registers ... 461
Registers - Description ... 462
**10.14 Sorting data ... 465**
**10.15 Modbus/TCP ... 466**
**10.15.1 Modbus/TCP server ... 467**
Addressing ... 468
Supported commands - Class 0 ... 470
Supported commands - Class 1 ... 471
Supported commands - Class 2 ... 472

**10.15.2 Modbus/TCP client ... 473**
**10.15.3 Modbus/TCP client with STX variables ... 475**
**10.16 User-programmable serial interface ... 477**
**10.16.1 Interface ... 478**
Serial interface port X11 ... 479
**10.16.2 Functioning principle of the user-programmable serial interface ... 482**
Functioning principle ... 483
**10.16.3 Registers ... 486**
Register numbers ... 487
Registers - Description ... 488
**10.16.4 Programming ... 495**
Configuring the interface ... 496
Sending characters ... 497
Sending texts ... 498
Sending values ... 499
Receiving characters ... 500
Receiving values ... 501
**10.17 User-programmable IP interface ... 502**
**10.17.1 Programming ... 504**
Initializing the user-programmable IP interface ... 505
Establishing a connection ... 506
Sending data ... 510
Receiving data ... 512
Terminating a connection ... 515
**10.17.2 Registers ... 516**
Register numbers ... 517
Register description ... 518
**10.18 CANopen® STX API ... 521**
STX function: CanOpenInit() ... 525
STX function: CanOpenSetCommand() ... 527
STX function: CanOpenUploadSDO() ... 529
STX function: CanOpenDownloadSDO() ... 532
STX function: CanOpenAddPDORx() ... 535
STX function: CanOpenAddPDOTx() ... 540
Heartbeat monitoring ... 544
CANopen® object dictionary ... 548
**10.19 User-programmable CAN-Prim interface ... 552**
Restrictions regarding the CAN-Prim interface ... 553
User-programmable CAN-Prim interface - Operating principle ... 557
Internal processes of the CAN-Prim interface ... 558
Register description - CAN-Prim interface ... 559
CAN message box - Description of registers for direct access ... 564
CAN message box - Description of registers for indirect access ... 570
Using the CAN-Prim interface ... 574
Using CAN-ID masks ... 577
RTR frames via CAN-Prim interface ... 578

**11 Automatic copying of controller data 580**

**11.1 Operating principle ... 582**
Activating the AutoCopy feature ... 583
Executing AutoCopy commands ... 584
Terminating AutoCopy mode ... 586
**11.2 autocopy.ini - Structure ... 587**
Section [OPTIONS] ... 588
Command sections ... 589

Example of a command file ... 597
**11.3 Log file ... 600**
File contents ... 601
**11.4 Data files ... 602**
File format ... 603

**12 OS update 605**

**12.1 Updating the operating system of the controller ... 606**
OS update by means of JetSym ... 607
Operating system update via FTP ... 608
Automatic OS update from an SD card ... 609
Operating system update from within the application program ... 610
**12.2 OS update of a JX module ... 611**
OS update by means of JetSym ... 612
Operating system update via FTP ... 613
Automatic OS update from an SD card ... 614
Operating system update from within the application program ... 615

**13 Application program 617**

Application program - Default path ... 618
The application program is stored to the SD card ... 619
Loading an application program ... 620

**14 Quick reference - JC-3xx 621**

**Appendix 637**

**A: Technical specifications ... 638**
JC-360: Technical data ... 639
Physical dimensions ... 641
Operating parameters - Environment and mechanics ... 642
Operating parameters: Enclosure ... 643
DC power supply inputs and outputs ... 644
Shielded data and I/O lines ... 645
**B: Index ... 646**

# 1 Safety instructions

**Introduction**

This chapter informs the user of basic safety instructions. It also warns the user of residual dangers, if there are any. Furthermore, it contains information on EMC.

**Contents**


| Topic | Page |
|---|---|
| Basic safety instructions | 14 |
| Instructions on EMI | 16 |

## Basic safety instructions

**Introduction**

This device complies with the valid safety regulations and standards. Jetter AG attaches great importance to the safety of the users.

Of course, the user should adhere to the following regulations:

- Relevant accident prevention regulations
- Accepted safety rules
- EC guidelines and other country-specific regulations

**Intended conditions of use**

Usage according to the intended conditions of use implies operation in accordance with this User Manual.

The controller JC-360 is used to control machinery, such as conveyors, production machines, and handling machines.

Operate the controller JC-360 only within the limits and conditions set forth in the technical specifications. The operating voltage of the controller JC-360 is classified as SELV (Safety Extra Low Voltage). Therefore, the JC-360 controller is not subject to the EU Low Voltage Directive.

**Usage other than intended**

The device must not be used in technical systems which to a high degree have to be fail-safe, such as, for example, in ropeways and airplanes.

The JC-360 is no safety-related part as per Machinery Directive 2006/42/EC. This device is not qualified for safety-relevant applications and must, therefore, NOT be used to protect persons.

If you intend to operate the device at ambient conditions not being in conformity with the permitted operating conditions, please contact Jetter AG beforehand.

**Personnel qualification**

Depending on the life cycle of the product, the persons involved must possess different qualifications. These qualifications are required to ensure proper handling of the device in the corresponding life cycle.

| Product life cycle | Minimum qualification |
|---|---|
| **Transport/storage:** | Trained and instructed personnel with knowledge in handling electrostatic sensitive components. |
| **Mounting/installation:** | Specialized personnel with training in electrical engineering, such as industrial electronics technician. |
| **Commissioning/ programming:** | Trained and instructed experts with profound knowledge of, and experience with, electrical/drive engineering, such as electronics engineer for automation technology. |
| **Operation:** | Trained, instructed and assigned personnel with knowledge in operating electronic devices. |
| **Decommissioning/ disposal:** | Specialized personnel with training in electrical engineering, such as industrial electronics technician. |

**Modifications and alterations to the module**

**For safety reasons, no modifications and changes to the device and its functions are permitted.**

Any modifications to the device not expressly authorized by Jetter AG will

| | |
|---|---|
| | result in a loss of any liability claims to Jetter AG.<br>**The original parts are specifically designed for the device. Parts and equipment from other manufacturers have not been tested by Jetter AG and are, therefore, not released by Jetter AG.**<br>The installation of such parts may impair the safety and the proper functioning of the device.<br>Any liability on the part of Jetter AG for any damages resulting from the use of non-original parts and equipment is excluded. |
| **Transport** | The JC-360 contains electrostatically sensitive components which can be damaged if not handled properly.<br>To exclude damages to the JC-360 during transport, the backplane module must be mounted to it, and it must be shipped in its original packaging, as well as in apt protective packaging..<br>▪ Use an appropriate outer packaging to protect the JC-360 against impact or shock.<br>▪ In case of damaged packaging inspect the device for any visible damage. Inform your freight forwarder and Jetter AG. |
| **Storing** | When storing the JC-360 observe the environmental conditions given in the technical specification. |
| **Repair and maintenance** | The operator is not allowed to repair the device. The device does not contain any parts that could be repaired by the operator.<br>If the device needs repairing, please send it to Jetter AG. |
| **Replacing modules** | When replacing the JC-360, class of protection IP20 is not ensured. Do not touch any electronic components once a module housing has been removed from the backplane module.<br>If you touch the EMC clip, you may damage this clip. A damaged clip may result in lower noise immunity. |
| **Disposal** | When disposing of devices, the local environmental regulations must be complied with. |

## Instructions on EMI

**Noise immunity of a system**

The noise immunity of a system is determined by the weakest component of the system. For this reason, correct wiring and shielding of cables is of paramount importance.

**Measures**

Measures for increasing EMI in electric plants:

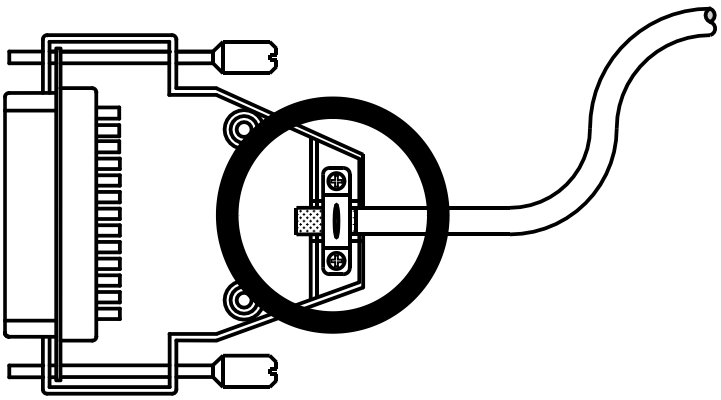
- Attach the JC-360 to a DIN rail to EN 60715 having got the dimensions 35 x 7.5 mm.
  The DIN rail must be electrically conducting and grounded by either of the two ways:
  - Directly:
  - Via rear panel of the electric cabinet
- Also refer to Application Note 016 *EMC-compatible installation of electric cabinets* by Jetter AG.

**The following instructions are excerpts from Application Note 016:**

- **Physically separate** signal and power lines. Jetter AG recommend spacing greater than 20 cm. Cables and lines should cross each other at an angle of 90°.
- The following line cables must be shielded:
  Analog lines, data lines, motor cables coming from inverter drives (servo output stage, frequency converter), lines between components and interference suppressor filter, if the suppressor filter has not been placed at the component directly.
- Shield cables **at both ends**.
- Unshielded wire ends of shielded cables should be as short as possible.
- The entire shield, **must, in its entire perimeter,** be drawn behind the isolation, and then be clamped under the earthed strain relief **with the greatest possible surface area.**

**When male connectors are used:**

- Draw the shield, in its entire perimeter, under the shielding clamp of the metallized connector housing (impedance shielding), respectively of the EMC gland bushing, its greatest possible surface area being clamped under a strain relief.
- Only use metallized connectors, e.g. Sub-D with metallized housing. Make sure that the strain relief is directly connected with the housing here as well.



**Downloading Application Note 016**

You can download Application Note 016 *EMC-Compatible Installation of Electric Cabinets* from the Jetter AG **homepage http://www.jetter.de**. In order to download Application Note 016, browse the following path: *Industrial Automation - Support - Downloads - 07_application_notes*.

# 2 Product description and design

**Introduction**

This chapter covers the design of the device, as well as how the order reference is made up including all options.

**Contents**


| Topic | Page |
|---|---|
| Product description of the JC-360 | 20 |
| Parts and interfaces of the controller JC-360 | 21 |
| Order reference/options | 22 |
| List of documentation | 23 |
| Accessories for the JX3 system | 25 |
| Physical dimensions | 26 |

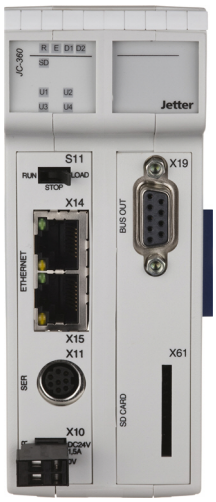
# Product description of the JC-360

**The controller JC-360**

The JC-360 is a high-performance compact controller. It covers all areas of industrial automation.

**Product features**

The features of this product are listed below:




- 4, 8 or an unlimited number of axes (on the CAN bus: 16 axes max.)
- 2 Ethernet ports with integrated switch
- Powerful programming language JetSym STX
- Non-volatile registers: 60,000 (option -R: 120,000)
- Program/data memory: 4 MB
- 1 serial port (RS-232/422/485)
- 1 JX2 system bus port
- 1 CAN port (CANopen®)
- Up to 16 JX3 modules can directly be added.
- Real-time clock
- Modbus/TCP
- SD memory card

**Additional options**

You must already specify the additional options for your controller when placing the order. The controller cannot be equipped with additional features afterwards. When ordering the controller JC-360 you may choose the following options:

- Additional memory (non-volatile registers: 120,000)

**Scope of delivery**

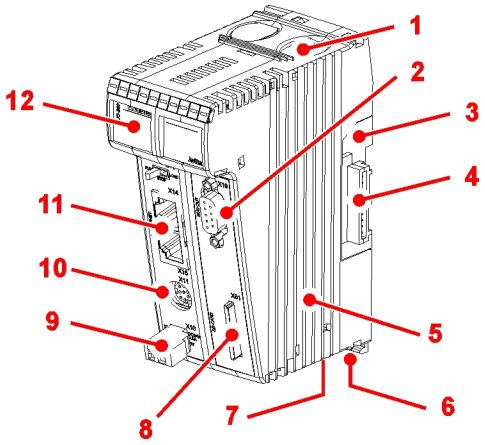
The following items are included in the scope of delivery of the JC-360:

| Item no. | Quantity | Description |
|---|---|---|
| Depending on options | 1 | Controller JC-360 |
| 60870409 | 1 | 2-pin connector, spring-cage connection |
| 60870411 | 10 | Terminal labels |
| 60873051 | 1 | Installation manual |
| 60870410 | 1 | Keying pins |

# Parts and interfaces of the controller JC-360

**Parts and interfaces**

The controller JC-360 features the following parts and interfaces:




| Number | Part | Description |
|---|---|---|
| **1** | Upper latch | Lets you remove the module enclosure from the backplane module |
| **2** | X18 | JX2 system bus interface |
| **3** | Backplane module | For installing the module on a DIN rail |
| **4** | X119 | Connector for additional JX3 modules |
| **5** | Module enclosure | |
| **6** | DIN rail latch | For removing the JC-3xx from the DIN rail |
| **7** | Lower latch | Lets you remove the module enclosure from the backplane module<br>Not visible in the illustration |
| **8** | X61 | SD card slot |
| **9** | X10 | Power supply |
| **10** | X11 | Serial interface |
| **11** | X14, X15 | Two Ethernet ports |
| **12** | LED | Diagnostic and status LEDs |

## Order reference/options

**Order reference**

The order reference consists of the name of the JC-360 and the desired options. Each of the additional options given below supplements the JC-360. The order reference only reflects existing options.

| JC-360 | - | A | - | R |
|---|---|---|---|---|

| Element | Description |
|---|---|
| JC-360 | Controller |
| A | Number of axes: 4, 8 or unlimited;<br>if option "unlimited" is chosen, A is not applicable, e.g. JC-360-R |
| R | Additional memory option: 120,000 non-volatile registers |

**Ordering additional options**

Specify your desired options in the order. The controller cannot be equipped with additional features afterwards.

**Number of axes**

Four, eight or an unlimited number of axes can be connected to the JC-360.

| Item no. | Order reference |
|---|---|
| 10000656 | JC-360-4 |
| 10000657 | JC-360-8 |
| 10000676 | JC-360 |

**Integrated Web server and e-mail feature**

The integrated Web server and e-mail feature of the JC-360 support the following functions by default:

- **HTTP server**: This feature lets the user download the homepages into the controller via FTP.
- **SMTP client**: This feature lets the controller send e-mails.

**Modbus/TCP**

The JC-360 supports the Modbus/TCP protocol. The controller can act as both server and client.

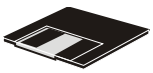
# List of documentation

**Introduction**

Various documents and software tools support you in engineering, installing and programming the JC-360 controller. You can download these documents and software tools from the Jetter AG **homepage http://www.jetter.de**.

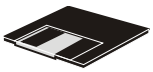
**Engineering**

The following documents and files support you in engineering the controller:

| | |
|---|---|
|  | **Industrial automation catalog** |
| | ■ Product description |
| | ■ Technical specifications |
|  | **Manual on the controller JC-3xx** |
| | ■ The document at hand |
|  | **CAD data of the controller JC-360** |
| | ■ dxf file with 2D illustrations |
| | ■ stp file with 3D illustrations |

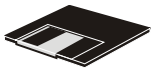
**Engineering a JX2 station on the JX2 system bus**

The following document and the following software tool support you in engineering a JX2 station on the JX2 system bus:

| | |
|---|---|
|  | **Manual on the controller JC-3xx** |
| | ■ System bus topology |
| | ■ JX2 system bus specification |
| | ■ Product descriptions of JX2 modules, IP 67 modules, as well as third-party modules |
|  | **System bus configurator** |
| | ■ Excel file for designing the system bus |
| | ■ SysBus_Configuration_xxx_e.xls (xxx: Version) |

**Engineering a JX3 station on the JX3 system bus**

The following document and the following software tool support you in engineering a JX3 station on the JX3 system bus:

| | |
|---|---|
|  | **Manual on the controller JC-3xx** |
| | ■ Engineering a JX3 station |
| | ■ Product descriptions of JX3 modules |
|  | **System bus configurator** |
| | ■ Excel file for designing the JX3 system bus |
| | ■ JX3-SysBus_Configurator_xxx_e.xls (xxx: Version) |

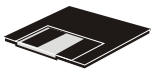
**Installation**

The following documents support you at installing the controller:

| | |
|---|---|
|  | **Installation manual** |
| | It is included in the boxed controller JC-360 and contains information on: |
| | ■ Installing the controller on a DIN rail |
| | ■ Terminal assignment |
| | ■ Specification of terminals |
| | ■ Diagnostics via LEDs |
|  | **Manual on the controller JC-3xx** |
| | ■ The document at hand |

**Programming**

The following document and software tool support you at programming the controller:

| | |
|---|---|
|  | **Manual on the controller JC-3xx** |
| | ■ The document at hand |
|  | **JetSym** |
| | ■ Programming Tool |

# Accessories for the JX3 system

**Labelling strips**

Ten labelling strips are included in the scope of delivery of the JC-360.



| | |
|---|---|
| Order reference | DIV_DEK_5/5_MC-10_NEUT_WS |
| Item no. | 60870411 |
| Packaging unit | 100 pcs. |

**Keying pins**



| | |
|---|---|
| Order reference | DIV_BL_SL_3.5_KO_OR |
| Item no. | 60870410 |

**Strain relief for BU_10_E_BLZF_GE_RM 3.5**



| | |
|---|---|
| Order reference | DIV_BL_3.5_ZE_8 |
| Item no. | 60870963 |

**End clamp for DIN rail**



| | |
|---|---|
| Order reference | DIV_CLIPFIX_35 |
| Item no. | 60863970 |

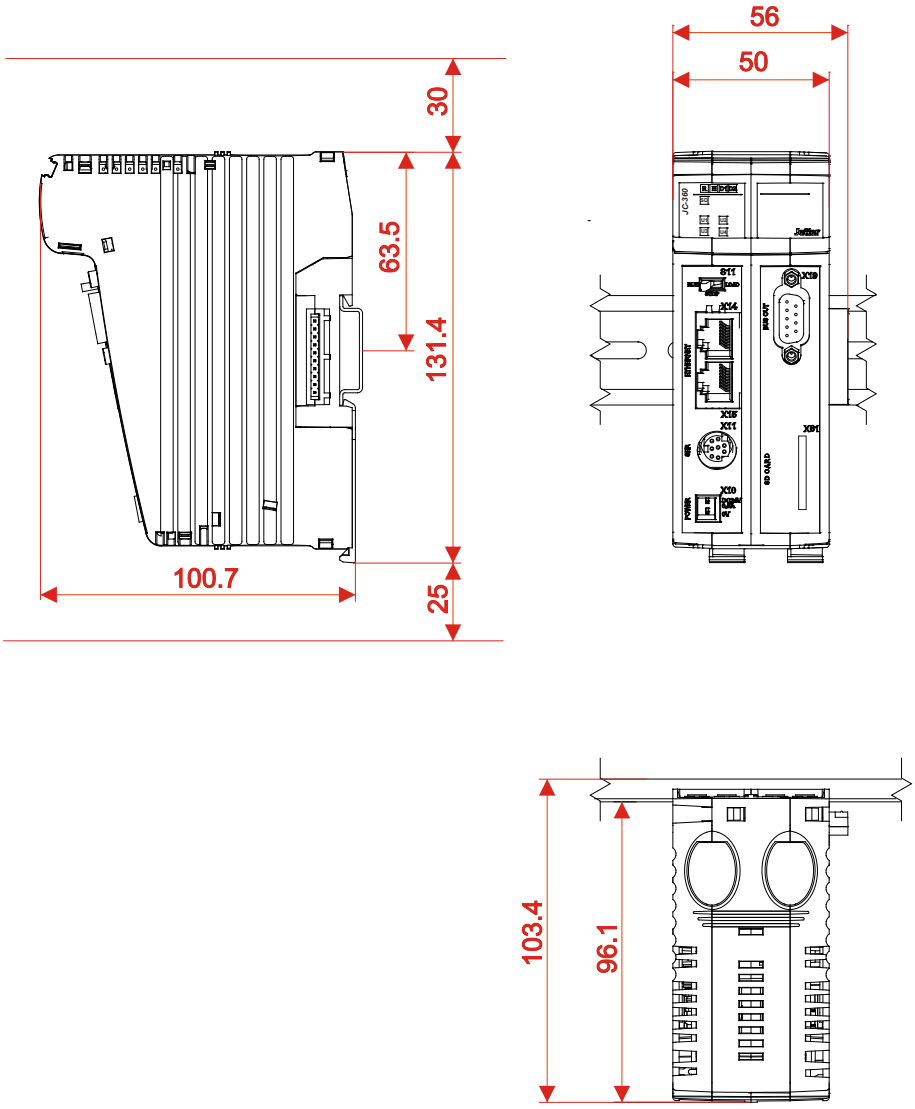
**Screwdriver**



| | |
|---|---|
| Type | SD 0.4 x 2.5 - DIN 5264-A |
| Order reference | DIV_SCHRAUBENDREHER_2,5*75 |
| Item no. | 60871712 |

## Physical dimensions

### Physical dimensions




| | |
|---|---|
| **Minimum clearances** | At mounting the JC-360, keep the minimum clearance above and below the controller. This way, there must be enough room to press the latches of the JX3 backplane module when replacing modules.<br>▪ Minimum clearance, above: 30 mm<br>▪ Minimum clearance, below: 25 mm |
| **Module width** | The width of the JC-360 is 56 mm. When the JC-360 is attached to a JX3 station, its width increases by 50 mm. |
| **Mounting orientation** | The orientation of the JC-360 is vertical. |

# 3 Identifying

**Purpose of this chapter**

This chapter is for supporting you in identifying the following information with regard to JC-360:

- Determining the hardware revision
- Retrieving Electronic Data Sheet (EDS) information. The EDS holds numerous remanent production-relevant data.
- Determining the OS version of the device and its software components

**Prerequisites**

To be able to identify the JC-360, the following prerequisites must be fulfilled:

- The controller is connected to a PC.
- The programming software JetSym Identifikation ab JetSym-Version or higher is installed on the PC.

**Information for hotline requests**

If you wish to contact the hotline of Jetter AG in case of a problem, please have the following information on the JC-360 ready:

- Serial number
- OS version number
- Hardware revision

**Contents**


| Topic | Page |
|---|---|
| Identification by means of the nameplate | 28 |
| Electronic Data Sheet EDS | 30 |
| Version registers | 37 |

# 3.1 Identification by means of the nameplate

**Introduction**

The nameplate is attached to the housing of the JC-360 and contains details, such as hardware revision number and serial number. If you wish to contact the hotline of Jetter AG in case of a problem, please have this information ready.

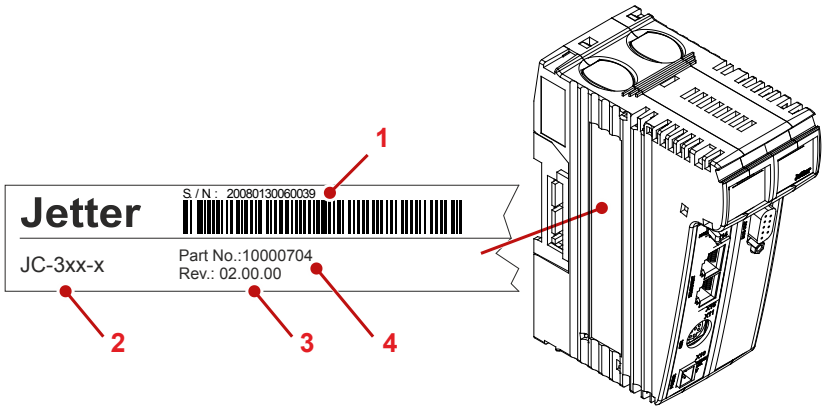
**Contents**


| Topic | Page |
|---|---|
| JC-3xx: Nameplate | 29 |

## JC-3xx: Nameplate

**Nameplate**

The nameplate of a JC-3xx controller contains the following information:




| Number | Description |
|---|---|
| **1** | Serial number |
| **2** | Controller name |
| **3** | Hardware revision |
| **4** | Item number |

# 3.2 Electronic Data Sheet EDS

**Introduction**

Each JC-360 features an Electronic Data Sheet (EDS). Numerous production-relevant data are permanently stored in the EDS. The EDS data can be read out via files in the file system of the JC-360 or via special registers.

**Contents**


| Topic | Page |
|---|---|
| EDS file | 31 |
| EDS registers | 35 |

# EDS file

**Introduction**

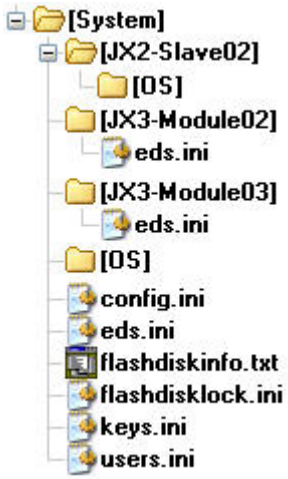
EDS data can be retrieved from the file **eds.ini**.

**Properties**

- You can access this file through the file system of the controller.
- For an FTP connection, the user needs administrator rights (user *admin*) or system rights (user *system*).
- The EDS file of the controller is located in the folder **System**.
- The EDS file of JX3 modules is located in the directory of the corresponding module */System/JX3-ModuleXX*.
- This file is read-only.
- Formatting the flash disk or SD card does not influence this file.

**Path to EDS files**

The illustration below shows an example of the contents of the directory **System** holding the EDS files of the controller and JX3 modules:




In the directory **JX3-ModuleXX** of the JX3 modules, the OS of which can be updated by transferring an OS file ***.os**, there is the directory **OS** which is not shown in the image.

**File structure**

The EDS file is a text file the entries of which are grouped into several sections.

**Example**

This is an example of an EDS file belonging to a JetControl 360:

```
;Jetter AG Electronic Data Sheet

[IDENTIFICATION]
Version = 2
Code = 864
Name = JC-360-8-R
PcbRev = 03
PcbOpt = 03
OSVersionMin = 1.13.0.0
BLVersionMin = 9.2.0.0

[PRODUCTION]
Version = 0
SerNum = 20150602000000
Day = 2
Month = 6
Year = 2015
TestNum = -1
TestRev = 255,255,255,255

[FEATURES]
Version = 1
MAC-Addr = 00:50:CB:00:00:00
Serial = 1
Switch = 1
STX = 1
NVRegs = 60000
JX3 bus = 1
CAN = 1
SD card = 1
Axes = 8
Web = 1
ModbusTCP = 1
SDLed = 1
UserLeds = 1
RTC = 1
```

**Section [IDENTIFICATION]**

The general hardware configuration can be seen from section [IDENTIFICATION].

| Name | Example | Function |
|---|---|---|
| Version | 2 | Version of this section |
| Code | 864 | Module code for JC-360 |
| Name | JC-360-8-R | Corresponds to the information on the nameplate |
| PcbRev | 03 | Hardware revision |

| Name | Example | Function |
|---|---|---|
| PcbOpt | 03 | Hardware option |
| OSVersionMin | 1.13.0.0 | The product is available as of this os version |
| BLVersionMin | 9.2.0.0 | The product is available as of this bootloader version |

**Section [PRODUCTION]**

The serial number and production date can be seen from section [PRODUCTION].

| Name | Example | Function |
|---|---|---|
| Version | 0 | Version of this section |
| SerNum | 20150602000000 | Corresponds to the information on the nameplate |
| Day | 02 | Production date: Day |
| Month | 06 | Production date: Month |
| Year | 2015 | Production date: Year |
| TestNum | -1 | Internal usage |
| TestRev | 255,255,255,255 | Internal usage |

**Section [FEATURES]**

In the section [FEATURES] special properties of the controller are specified. The OS of the controller will ignore properties of missing entries in the file.

| Name | Example | Function |
|---|---|---|
| Version | 1 | Version of this section |
| MAC Addr | 00:50:CB:00:00:00 | Ethernet MAC address |
| Serial | 1 | The serial interface is available |
| Switch | 1 | A switch (mode selector) RUN/STOP/LOAD is available |
| STX | 1 | Runtime environment for the application program is available |
| NVRegs | 60000 | Number of non-volatile registers |
| JX3 bus | 1 | Bus interface for JX3 modules is available |
| CAN | 1 | Bus interface for JX2 modules is available |
| SD card | 1 | Slot for the SD memory card is available |
| Axes | 8 | Number of supported JX2 axis modules |
| Web | 1 | Web server and e-mail client are available |
| ModbusTCP | 1 | Modbus/TCP client and server are available |
| SD LED | 1 | The LED for the SD memory card is available |
| UserLEDs | 1 | User-specific LEDs are available |
| RTC | 1 | A real-time clock is available |

**Example: JX3 modules**

Examples of EDS files for JX3 modules: Please refer to the manual of the corresponding module.

**Related topics**

- **EDS registers** (see page 35)

# EDS registers

**Introduction**

EDS registers let you retrieve entries in the Electronic Data Sheet (EDS).

**Register numbers**

The basic register number is dependent on the controller. The register number is calculated by adding the number of the module register (MR) to the number of the basic register.

| Device | Basic register number | Register numbers |
|---|---|---|
| JC-360 | 100000 | 100500 ... 100817 |

**Readable data**

The following table lists the EDS registers of a controller, as well as their connection to the entries in the EDS file **/System/eds.ini**. By means of this register array, you can have the EDS of the controller or of a JX3 module displayed. For this, you must select the controller or the desired JX3 module via module registers 500 and 501. The contents of the selected EDS are then displayed in the following registers.

| Register | Section in the EDS file | Name in the EDS file | Description |
|---|---|---|---|
| **MR 500** | - | - | Functional group:<br>0: CPU<br>1: JX3 modules |
| **MR 501** | - | - | Module number<br>(if MR 500 > 0) |
| **MR 600** | IDENTIFICATION | Version | Version of this section |
| **MR 601** | | Code | Module code |
| **MR 602 through MR 612** | | Name | Module name or controller name |
| **MR 613** | | PcbRev | Hardware revision |
| **MR 614** | | PcbOpt | Hardware option |
| **MR 700** | PRODUCTION | Version | Version of this section |
| **MR 701 through MR 707** | | SerNum | Serial number |
| **MR 708** | | Day | Production date: Day |
| **MR 709** | | Month | Production date: Month |
| **MR 710** | | Year | Production date: Year |
| **MR 711** | | TestNum | Internal usage |
| **MR 712** | | TestRev | Internal usage |

| Register | Section in the EDS file | Name in the EDS file | Description |
|---|---|---|---|
| **MR 800** | FEATURES | Version | Version of this section |
| **MR 801** | | MAC Addr | MAC address (manufacturer section) |
| **MR 802** | | MAC Addr | MAC address (device section) |
| **MR 803** | | Serial | Serial interface |
| **MR 804** | | Switch | Mode selector RUN/STOP/LOAD |
| **MR 805** | | STX | Runtime environment for the application program |
| **MR 806** | | NVRegs | Number of remanent registers |
| **MR 807** | | JX3 bus | Bus interface for JX3 modules |
| **MR 808** | | CAN | Bus interface for JX2 modules |
| **MR 809** | | SD card | SD card slot |
| **MR 810** | | MotionControl | MC software |
| **MR 811** | | Axes | Number of supported JX2 axis modules |
| **MR 812** | | Web | Web server and e-mail client |
| **MR 813** | | ModbusTCP | Modbus/TCP client and server |
| **MR 815** | | SD LED | LED of the SD card slot |
| **MR 816** | | UserLEDs | User-defined LEDs |
| **MR 817** | | RTC | Real-time clock |

**EDS file of JX3 modules**

The assignment of module registers 6xx and 7xx corresponds to the assignment with JX3 module.

**Related topics**

- **EDS file** (see page 31)

# 3.3 Version registers

**Introduction**

The operating system provides several registers which can be used to read out the revision number of the hardware or the version of the operating system and its components. If you wish to contact the hotline of Jetter AG in case of a problem, please have this revision ready.

**Contents**


| Topic | Page |
|---|---|
| Hardware revisions | 38 |
| Software versions | 39 |

# Hardware revisions

**Introduction**

The JC-360 features special registers, the content of which lets you identify the hardware.

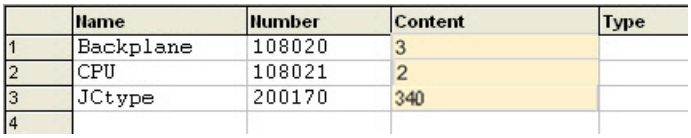
**Register overview**

The following registers let you read out the hardware revisions:

| Register | Description |
|---|---|
| **108020** | Hardware revision of the backplane module |
| **108021** | Hardware revision - CPU board |
| **200170** | Controller type |

**Version numbers in JetSym setup**

The following screenshot shows a JetSym setup window displaying the version registers:



| | Name | Number | Content | Type |
|---|---|---|---|---|
| 1 | Backplane | 108020 | 3 | |
| 2 | CPU | 108021 | 2 | |
| 3 | JCtype | 200170 | 340 | |
| 4 | | | | |

**Related topics**

- **Software versions** (see page 39)

# Software versions

**Introduction**

The JC-360 features software with unique version numbers which can be read out from special registers.

**Format of software version numbers**

The software version numbers of the JC-360 are four-figure values.

| 1 | . | 2 | . | 3 | . | 4 |
|---|---|---|---|---|---|---|

| Element | Description |
|---|---|
| **1** | Major or main version number |
| **2** | Minor or secondary version number |
| **3** | Branch or intermediate version number |
| **4** | Build version number |

**Released version**

A released version can be recognized by both Branch and Build having got value 0.

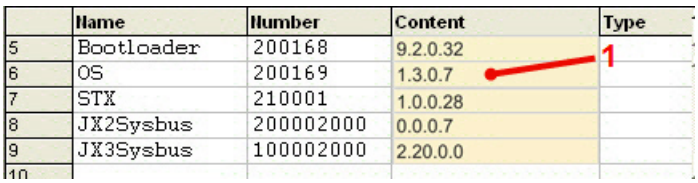
**Register overview**

The following registers let you read out the software versions:

| Register | Description |
|---|---|
| **200168** | Boot loader version |
| **200169** | Operating system version |
| **210001** | Version of the execution unit for the STX application program |
| **200002000** | Version of the JX2 system bus driver |
| **100002000** | Version of the JX3 system bus driver |

**Version numbers in JetSym setup**

The following screenshot shows a JetSym setup window displaying version registers. To have the version number displayed in the setup window of JetSym, select the format **IP address**.




| Number | Description | Function |
|---|---|---|
| **1** | V 1.03.0.07 | OS version number of the controller<br>JetSym displays this information in the title bar of each setup window. |

**Related topics**

- **Hardware revisions** (see page 38)

# 4 Mechanical and electrical installation

**Purpose of this chapter**

This chapter deals with mechanical and electrical installation of the JC-360 regarding the following aspects:

- Wiring the JC-360
- Description of the indicators
- Description of control elements
- Mechanical installation
- Connecting JX3 modules to the JC-360
- Connecting modules to the JX2 system bus
- Connecting displays and HMIs

**Contents**


| Topic | Page |
|---|---|
| Interfaces | 42 |
| JX2 system bus interface - Female connector X19 | 48 |
| LEDs indicating various states | 54 |
| Controls and SD memory card | 61 |
| Installing, replacing and removing the module | 66 |
| IP configuration | 71 |
| Engineering of JX3 station equipped with a JC-360 | 91 |
| Configuring the JX2 system bus | 104 |
| Connecting displays and HMIs | 130 |

# 4.1 Interfaces

**Terminal X10**

The function of terminal X10 is as follows:

- Power supply for the controller JC-360
- Power supply for connected JX3 peripheral modules

**Terminal X11**

The function of terminal X11 is as follows:

- Serial interface to a PC
- Serial interface to an HMI by Jetter AG
- Serial interface to any device

**Connectors X14, X15**

The function of terminals X14 and X15 is as follows:

- Ethernet port to a hub, switch or router
- Ethernet port to a PC
- Ethernet port to an HMI by Jetter AG
- Ethernet port to a JX3-BN-ETH, a JX3-COM-xxxx, or a JetMove-200-ETH
- Ethernet port to any device

**Contents**


| Topic | Page |
|---|---|
| Power supply terminal X10 | 43 |
| Serial interface port X11 | 44 |
| Ethernet ports - Female connectors X14, X15 | 47 |

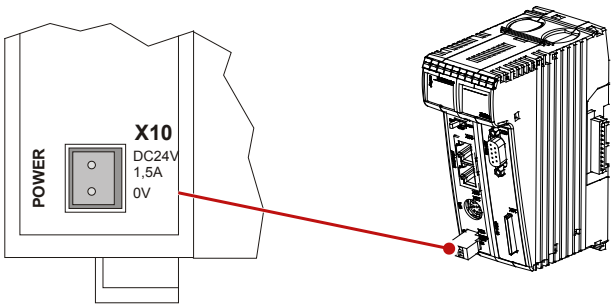
# Power supply terminal X10

**Devices to connect with this terminal**

X10 lets you connect the following devices:

- Power supply for the controller JC-360
- Power supply of JX3 modules which are connected upstream of a JX3-PS1.

**Terminal assignment**



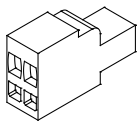

| Terminal point | Description |
|---|---|
| **DC 24 V 1.5A** | Power supply for the JC-360 and connected JX3 peripheral modules |
| **0 V** | Reference potential |

**Technical specifications**

| Parameter | Description |
|---|---|
| Rated voltage | DC 24 V |
| Permissible voltage range | -15 % ... +20 % |
| Input current without HMI | 1.0 A max. |
| Power consumption | 24 W max. |

**Connector for power supply terminal X10**

A 2-pin connector is included in the scope of delivery of the JC-360.



| | |
|---|---|
| Designation | BU_02_E_BLZF_GE_RM3.5 |
| Item no. | 60870409 |
| Connector technology | Spring-cage connection |
| Wire size: | 0.2 ... 1.5 $mm^2$(AWG 25 ... 14) |

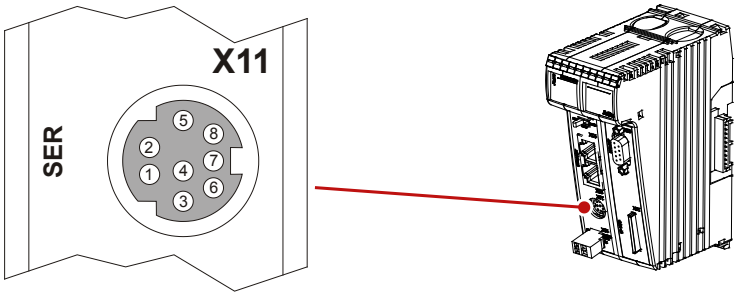
## Serial interface port X11

**Devices to connect with this port**

Port X11 lets you connect the following devices:

- A PC
- An HMI by Jetter AG
- Any device

**Pin assignment of port X11**




| Pin | Signal | Description |
|---|---|---|
| **1** | RDA | RS-422; receive data inverted |
| **2** | GND | Reference potential |
| **3** | RDB | RS-422; receive data not inverted |
| **4** | RxD | RS-232; receive data |
| **5** | SDB | RS-422; transmit data not inverted<br>RS-485; transmit/receive data not inverted |
| **6** | DC 24 V | HMI supply voltage |
| **7** | SDA | RS-422; transmit data inverted<br>RS-485; transmit/receive data inverted |
| **8** | TxD | RS-232; transmit data |

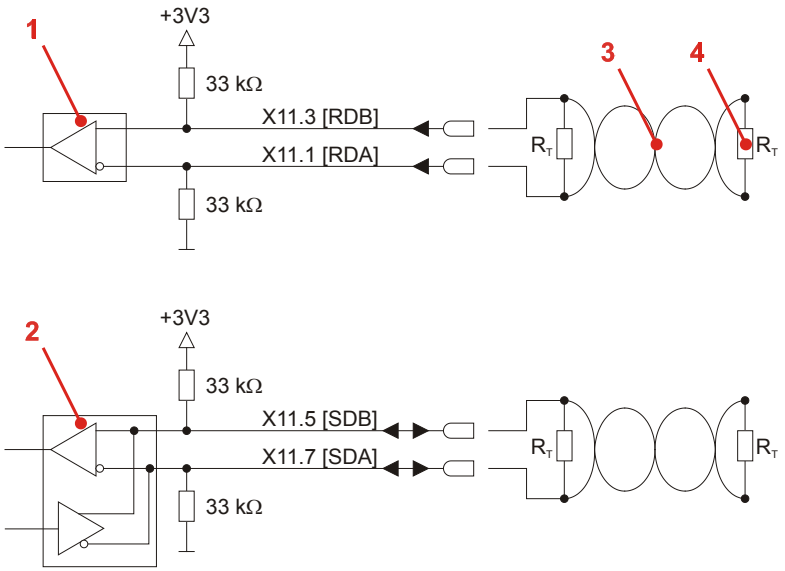
**Restrictions**

Irrespective of the fact that various hardware drivers have been implemented, only one hardware interface is available.

This means:

While, for example, communication via RS-422 is taking place, simultaneous and independent communication via RS-232 is not possible.

## Block diagram




| Number | Part | Function in the case of RS-422 | Function in the case of RS-485 |
|---|---|---|---|
| **1** | Receiver | Receives data | Unused |
| **2** | Receiver/transmitter | Transmits data | Receives and transmits data |
| **3** | Serial line | Twisted line of the serial interface | |
| **4** | $R_T$ | Terminating resistor | |

## Terminating resistor

Connect a terminating resistor to both serial lines in the following cases:

- Long lines
- High baud rates

Select a terminating resistor which corresponds to the impedance of the line used.

**Technical specifications**

| Parameter | Description |
|---|---|
| Type of terminal | MiniDIN, shielded |
| Number of pins | 8 |
| Electrical isolation | None |
| Number of interfaces | 1 serial interface |
| Interface standards | RS-232/RS-422/RS-485-2 |
| Baud rates | 2,400 ... 115,200 baud |
| Bits per character | 5, 6, 7, 8 |
| Number of stop bits | 1, 2 |
| Parity | Even, odd, none, 1, 0 |

**Cables for port X11**

For connecting devices to port X11 you can order the following cables:

| Item no. | Item | Description |
|---|---|---|
| 60867209 | KAY_0576-0050 | JetControl to modem with 9-pin Sub-D, length 0.5 m |
| 60868359 | Cable assy # 196 2.5M | JetControl to PC with 9-pin Sub-D, length 2.5 m |
| 60860013 | Cable assy # 196 5M | JetControl to PC with 9-pin Sub-D, length 5 m |
| 60868956 | Cable assy # 196 8M | JetControl to PC with 9-pin Sub-D, length 8 m |
| 60860011 | Cable assy # 192 2.5M | JetControl to HMI with 15-pin Sub-D, length 2.5 m |
| 60860012 | Cable assy # 193 5M | JetControl to HMI with 15-pin Sub-D, length 5 m |
| 60872142 | Cable assy # 192 10M | JetControl to HMI with 15-pin Sub-D, length 10 m |
| 60872884 | Cable assy # 192 15M | JetControl to HMI with 15-pin Sub-D, length 15 m |
| 60864359 | KAY_0386-0250 | JetControl to LCD 60 with 15-pin Sub-D, length 2.5 m |
| 60864360 | KAY_0386-0500 | JetControl to LCD 60 with 15-pin Sub-D, length 5 m |
| 60864897 | KAY_0533-0025 | JetControl to LCD 52/54 with 15-pin Sub-D, length 0.25 m |
| 60864257 | Cable assy # 197 5M | JetControl to JetView 200/300 with 9-pin Sub-D, length 5 m |
| 60871930 | Cable assy # 197 12M | JetControl to JetView 200/300 with 9-pin Sub-D, length 12 m |

# Ethernet ports - Female connectors X14, X15

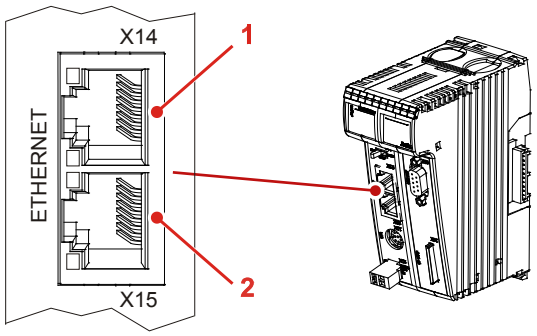
**Devices to connect with these ports**

Ports X14 and X15 let you connect the following devices:

- A PC
- An HMI by Jetter AG
- A JX3-BN-ETH, a JX3-COM-xxxx, or a JetMove-200-ETH
- Any device

**Pin assignment**

Ports X14 and X15 are internally connected via Ethernet switch.




| Number | Description |
|---|---|
| 1 | Connector X14 - Ethernet port |
| 2 | Connector X15 - Ethernet port |

**Technical specifications**

| Parameter | Description |
|---|---|
| Type of terminal | RJ45 Ethernet port |
| Number of ports | Two; one port per connector |
| Bit rate | 10 MBit/s, 100 MBit/s (Cat 5e) |
| Auto-crossover | Yes |

**Cables for ports X14 and X15**

For connecting devices to ports X14 and X15 you can order the following cables:

| Item no. | Item |
|---|---|
| 60537500 | Patch cable 1:1, 1 m gray Hirose, Cat 5e, shielded |
| 60854512 | Patch cable 1:1, 2 m gray Hirose, Cat 5e, shielded |
| 60854514 | Patch cable 1:1, 5 m gray Hirose, Cat 5e, shielded |
| 60854515 | Patch cable 1:1, 10 m gray Hirose, Cat 5e, shielded |

# 4.2 JX2 system bus interface - Female connector X19

**Introduction**

This chapter gives a description of the JX2 system bus interface of the JC-360.

**Connectable modules**

The JX2 system bus of the controller JC-360 lets you connect the following devices:

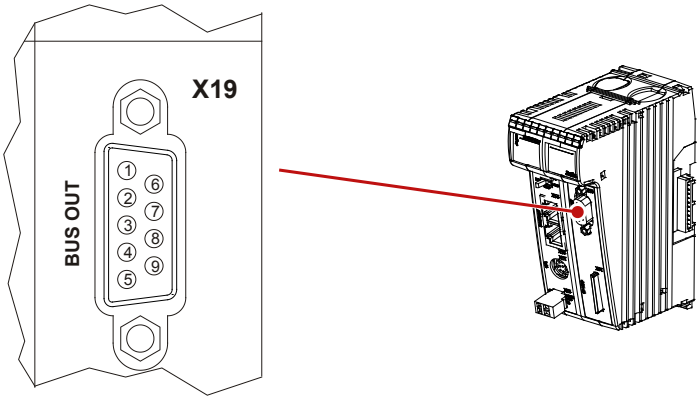
- JX2-I/O modules
- JX2 slave modules
- Servo amplifiers JetMove 1xx, JetMove 2xx, and JetMove 6xx
- IP67 modules LioN-S and LJX7-CSL
- Third-party CANopen® modules, e.g. valve terminals

**Contents**


| Topic | Page |
|---|---|
| Female connector X19 - Pin assignment | 49 |
| JX2 system bus cable specification | 51 |
| Line length and baud rate of the JX2 system bus | 53 |

# Female connector X19 - Pin assignment

## Pin assignment




| Pin | Signal | Description |
|---|---|---|
| **1** | CMODE0 | Commissioning |
| **2** | CAN-L | Data signal |
| **3** | GND | Reference potential |
| **4** | CMODE1 | Commissioning |
| **5** | Unused | |
| **6** | Unused | |
| **7** | CAN-H | Data signal |
| **8** | Unused | |
| **9** | Unused | |

## Technical specifications

| Parameter | Description |
|---|---|
| Type of terminal | Sub-D connector |
| Number of pins | 9 |
| Electrical isolation | None |
| Baud rates | 1,000/500/250/125 kBaud |

**Cable for connector X19**

For connecting modules to the JX2 system bus via connector X19 you can order the following cables:

| Item no. | Item |
|---|---|
| 10309001 | Cable assy # 530 0.2 m |
| 10309002 | Cable assy # 530 0.5 m |
| 10309003 | Cable assy # 530 1.0 m |
| 10309004 | Cable assy # 530 1.5 m |
| 10309006 | Cable assy # 530 2.0 m |
| 10309016 | Cable assy # 530 2.5 m |
| 10309015 | Cable assy # 530 3.0 m |
| 10309007 | Cable assy # 530 4.0 m |
| 10309008 | Cable assy # 530 5.0 m |

**Related topics**

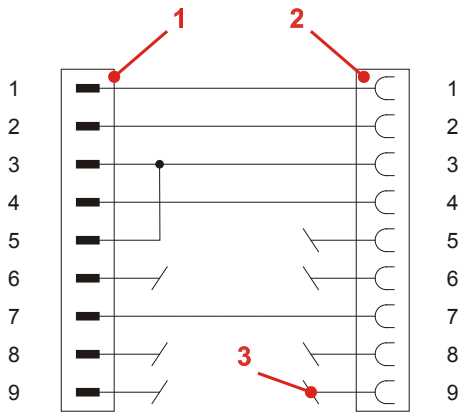
- **JX2 system bus cable - Specification** (see page 51)
- **Line length (in mm) and baud rate** (see page 53)

# JX2 system bus cable specification

### JX2 system bus cable specification

| Parameter | Description |
|---|---|
| Core cross-sectional area | 1,000 kBaud: 0.25 … 0.34 $mm^2$<br>500 kBaud: 0.34 … 0.50 $mm^2$<br>250 kBaud: 0.34 … 0.60 $mm^2$<br>125 kBaud: 0.50 … 0.60 $mm^2$ |
| Cable capacitance | 60 pF/m max. |
| Resistivity | 1,000 kBaud: 70 Ω/km max.<br>500 kBaud: 60 Ω/km max.<br>250 kBaud: 60 Ω/km max.<br>125 kBaud: 60 Ω/km max. |
| Number of cores | 5 |
| Shielding | Complete shielding, no paired shielding |
| Twisting | Core pairs CAN-L and CAN-H are twisted |

### Connection diagram




| Number | Part | Description |
|---|---|---|
| **1** | Male Sub-D connector, 9-pin | For connection to BUS-OUT |
| **2** | Female Sub-D connector, 9-pin | For connection to BUS-IN |
| **3** | Not connected | Do not connect these pins |

**Male Sub-D connector**

Pinout of the 9-pin male Sub connector at the JX2 system bus cable:

| Pin | Signal name | Description |
|---|---|---|
| **1** | CMODE0 | Commissioning |
| **2** | CAN-L | Data signal |
| **3** | GND | Reference potential |
| **4** | CMODE1 | Commissioning |
| **5** | TERM | Short-circuited with pin 3 |
| **7** | CAN-H | Data signal |

**Female Sub-D connector**

Pinout of the 9-pin female Sub-D connector to the JX2 system bus cable:

| Pin | Signal name | Description |
|---|---|---|
| **1** | CMODE0 | Commissioning |
| **2** | CAN-L | Data signal |
| **3** | GND | Reference potential |
| **4** | CMODE1 | Commissioning |
| **7** | CAN-H | Data signal |

# Line length and baud rate of the JX2 system bus

**Cable lengths**

The maximum cable length depends on the baud rate used and the number of expansion modules connected to the bus.

| Baud rate | Cable length | Stub length | Total stub length |
|---|---|---|---|
| 1,000 kBaud | 25 m max. | 0.3 m max. | 3 m |
| 500 kBaud | 100 m max. | 1.0 m max. | 39 m |
| 250 kBaud | 200 m max. | 3.0 m max. | 78 m |
| 125 kBaud | 200 m max. | - | - |

**Rules for calculating the stub length**

When engineering the line length, follow the rules listed below:

- Each non-intelligent JX2-I/O module connected to the system bus reduces the maximum line length by 1.0 m
- Each connected intelligent JX2-I/O slave module reduces the maximum line length by 1.0 m
- Each JetMove reduces the maximum line length by 1.0 m
- Each connected IP67-I/O module reduces the maximum line length by 1.0 m

**Baud rate**

The baud rate setting depends on the number of modules connected to the JX2 system bus:

| JX2-I/O modules<br>JX2 slave modules<br>JetMove | JX-SIO<br>CANopen® modules | 1,000 kBaud | 500 kBaud | 250 kBaud | 125 kBaud |
|---|---|---|---|---|---|
| x | | x | x | x | x |
| | x | x | x | x | x |
| x | x | x | | | x |

# 4.3 LEDs indicating various states

**LEDs of the JC-360**

The JC-360 features the following LEDs:

- 5 LEDs for indicating conditions and errors of the controller
- 4 LEDs for indicating user-defined events
- 4 LEDs for indicating the conditions of the Ethernet ports

**Contents**


| Topic | Page |
|---|---|
| LEDs of the controller | 55 |
| LEDs of the controller during boot process | 57 |
| LED indications on the controller after an IP address conflict has been detected | 59 |
| Status LEDs - Ethernet interface | 60 |

# LEDs of the controller

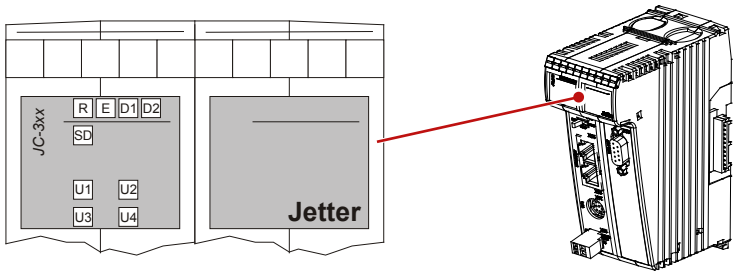
## LEDs of the controller

The controller JC-360 indicates conditions and errors via its LEDs. There are three groups of LEDs:

- Status messages of the operating system
- Application-specific LEDs
- Status of the Ethernet port

## OS- and application-specific LEDs

The status LEDs of the OS and application-specific LEDs are located in the diagnostic and status area below the labeling field.




| OS LED | Color | Description |
|---|---|---|
| **R** | Green | OS is running |
| **E** | Red | Generic error |
| **D1** | Red | Special conditions |
| **D2** | Red | Boot loader is running |
| **SD** | Amber | Access to SD card |

| Application-specific LED | Color | Description |
|---|---|---|
| **U1** | Amber | Programmable depending on the application |
| **U2** | Amber | Programmable depending on the application |
| **U3** | Amber | Programmable depending on the application |
| **U4** | Amber | Programmable depending on the application |

## Normal operating condition

In normal operating condition, the OS LEDs of the controller JC-360 indicate the following:

| R | E | D1 | D2 | SD | State |
|---|---|---|---|---|---|
| ON | OFF | OFF | OFF | OFF | Normal operating condition<br>■ Application program is running<br>■ No error<br>■ No access to SD card |

**States of the OS LEDs**

The table below shows the possible states of the OS-LEDs **R**, **E**, **D1** und **D2**:

| LED | State | Description |
|---|---|---|
| **R** | OFF | No power supply or failure |
| | 1Hz | Either the controller does not boot up, or it does not execute the application program. |
| | 4Hz | Reset or fatal error |
| | ON | Application program is being executed |
| **E** | OFF | No error |
| | 1Hz | No valid OS |
| | 4Hz | Reset, fatal error, or checking the network consistency |
| | ON | Error; refer to error register |
| **D1** | OFF | Normal operating condition |
| | 1Hz | Automatic IP configuration, or AutoCopy function is completed, or first part of startup delay is carried out. |
| | 4Hz | Reset or second part of startup delay is carried out, or fatal error has occurred. |
| | ON | File **autocopy.ini** on SD card is being executed. |
| **D2** | OFF | Boot loader is not running |
| | 1Hz | Boot loader: Automatic IP configuration |
| | 4Hz | Reset or fatal error |
| | ON | Boot loader is being executed |

# LEDs of the controller during boot process

**Normal boot process**

If the following requirements are met, the controller goes through its normal boot process:

- Mode selector S11 is in *RUN* position.
- There is a valid OS.
- There is a valid application program.

The LED **SD** is not shown here, as it is OFF during the normal boot process.

During boot process of the controller, the other OS status LEDs then indicate the following:

| Step | Description | | | | |
|---|---|---|---|---|---|
| **1** | | | | | |
| | **R** | **E** | **D1** | **D2** | **State** |
| |  4Hz |  4Hz |  4Hz |  4Hz | Reset |
| | | | | | |
| **2** | | | | | |
| | **R** | **E** | **D1** | **D2** | **State** |
| |  1Hz |  OFF |  OFF |  ON | Boot loader is running and is checking the OS |
| | | | | | |
| **3** | | | | | |
| | **R** | **E** | **D1** | **D2** | **State** |
| |  1Hz |  OFF |  OFF |  OFF | The OS reads out the DIP switch settings on the backplane module. Then, it checks if an Ethernet switch exists. |
| | | | | | |
| **4** | | | | | |
| | **R** | **E** | **D1** | **D2** | **State** |
| |  1Hz |  ON |  OFF |  OFF | The OS initializes real-time clock, Ethernet interface and file system. |
| | | | | | |
| **5a** | Only if in *start delay* register R 202971 a time value has been entered, the controller executes steps 5a and 5b. | | | | |
| | **R** | **E** | **D1** | **D2** | **State** |
| |  1Hz |  ON |  1Hz |  OFF | The first half of the start delay is in progress |
| | | | | | |

| | | | | | |
|---|---|---|---|---|---|
| **5b** | | | | | |
| | **R** | **E** | **D1** | **D2** | **State** |
| | 1Hz | ON | 4Hz | OFF | The second half of the start delay is in progress |
| | | | | | |
| **6** | | | | | |
| | **R** | **E** | **D1** | **D2** | **State** |
| | 1Hz | ON | ON | OFF | The OS initializes the modules on the JX3 and JX2 system bus, as well as software features (Web, Modbus/TCP, etc.); then it loads the application program. |
| | | | | | |
| **7** | | | | | |
| | **R** | **E** | **D1** | **D2** | **State** |
| | ON | OFF | OFF | OFF | Normal operating condition |
| | | | | | |

# LED indications on the controller after an IP address conflict has been detected

**IP conflict detected**

During booting or after the IP address has been modified through NetConsistency, the controller checks its own IP address. If the controller detects that its IP address is already used on the network, it takes the following actions:

- The controller sets bit 24 in R 200009.
- The controller sets bit 7 in R 200008. This is the collective error bit of R 200009.
- Then, the LEDs on the controller indicate the following:

| E | D1 | D2 | State |
|---|---|---|---|
| ON | 1Hz | 1Hz | The controller has detected an IP address conflict. |

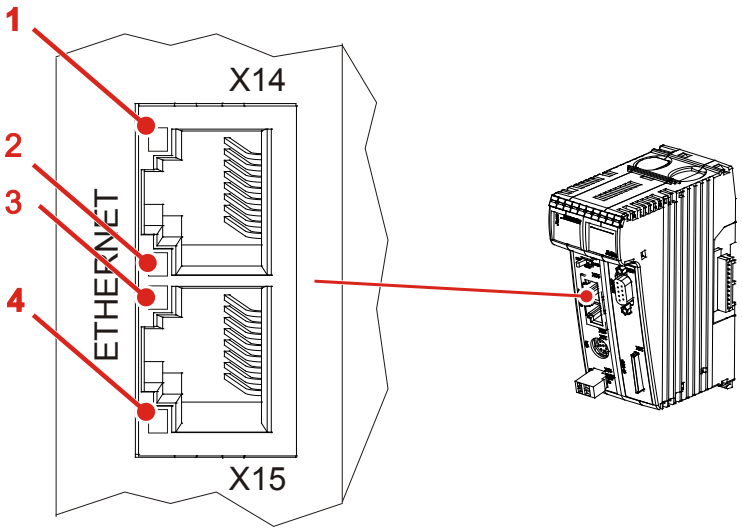
To exit this state, the following two steps must be taken:

| Step | Action |
|---|---|
| **1** | Resolve the IP address conflict. |
| **2** | Restart the controller. |

## Status LEDs - Ethernet interface

**Status LEDs - Ethernet interface**

The status LEDs of the Ethernet interface are located in the immediate vicinity of the RJ45 ports.




| LED | Color | Description |
|---|---|---|
| X14-1 | Green | LINK: Network connection has been established |
| X14-2 | Amber | ACT: Data transmission |
| X15-3 | Green | LINK: Network connection has been established |
| X15-4 | Amber | ACT: Data transmission |

# 4.4 Controls and SD memory card

**Control element of JC-360**

The JC-360 is equipped with the following controls:

- Mode selector S11 with the following positions: *RUN*, *STOP*, and *LOAD*

**SD card**

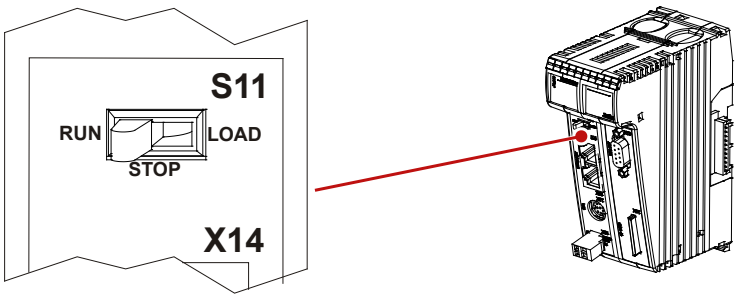
The JC-360 is equipped with a slot for SD memory cards. The SD card slot is an optional feature of the controller JC-340.

**Contents**


| Topic | Page |
|---|---|
| Function description of mode selector S11 | 62 |
| SD card slot X61 | 64 |

# Function description of mode selector S11

**Mode selector S11**




| Position | Behavior after power-up |
|---|---|
| **RUN** | The controller launches the application program. |
| **STOP** | The controller does not launch the application program. |
| **LOAD** | The controller executes the file **autocopy.ini** which is located on the SD card. |

**Functions of the mode selector**

The JC-360 checks the position of selector S11 in the following way:

| Step | Description | |
|---|---|---|
| **1** | Power supply of the controller is at terminal X10. | |
| **2** | The boot loader of the controller checks the position of selector S11. | |
| | **If ...** | **... then ...** |
| | ... mode selector S11 = *RUN* or *STOP*, ... | ... the OS is launched;<br>--> proceed with step 4. |
| | ... selector S11 = *LOAD* position, and an SD card has been inserted. | ... the OS is launched;<br>--> proceed with step 3. |
| | ... selector S11 = LOAD position, and an SD card has not been inserted. | ... the boot loader waits until an OS update is carried out |
| **3** | The controller loads the file**autocopy.ini**. | |
| | **If ...** | **... then ...** |
| | ... the file could be loaded, ... | ... the instructions contained in it are executed. |
| | ... the file could not be loaded, ... | ... proceed with step 4. |
| **4** | The controller checks the position of selector S11. | |
| | **If ...** | **... then ...** |
| | ... mode selector S11 = *RUN*, ... | ... the application program is launched. |
| | ... mode selector S11 = *STOP*, ... | ... the application program does not start. |

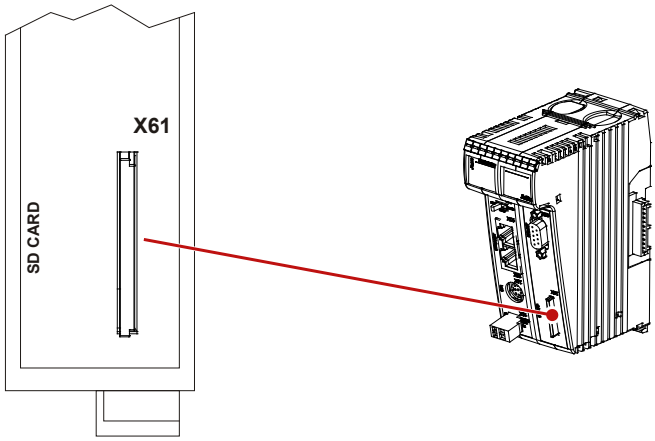
| Step | Description | |
|---|---|---|
| 5 | If ... | ... then ... |
| | ... the position of mode selector S11 is changed once the controller has been turned on, ... | ... this has no effect on the functioning of the controller. |

# SD card slot X61

**Introduction**

The SD card slot is for accommodating standard SD memory cards. The controller accesses data stored on the SD card which is used as file system extension. The SD card slot is an optional feature of the controller JC-340.

**SD card slot - Position**

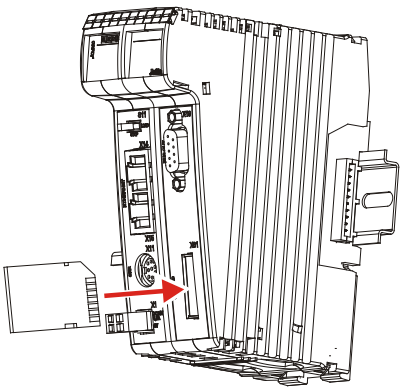



**Technical specifications**

| Parameter | Description |
|---|---|
| Plug-in card type | Standard SD card |
| Mechanical operation | push/push |
| Maximum memory size | 8 MB ... 4 GB |
| Card format | FAT |
| Protection against inserting the SD card in the wrong direction | Yes |

**Inserting the SD card**

Insert the SD card into the SD slot as illustrated below.



The way of inserting the SD card correctly is the same as in common digital cameras. If the SD card has been inserted correctly, the status LED **SD** lights up for 300 ms.

| | |
|---|---|
| **Removing the SD card** | Make sure the SD card is not accessed, while you are removing it. First, close all files which are stored to the SD card.<br>Remove the SD card the same way as you do it with digital cameras.<br>After removing the SD card, the status LED **SD** lights up twice for 100 ms each time. |

# 4.5 Installing, replacing and removing the module

**Introduction**

This chapter covers mounting, replacing and removing of controllers belonging to the JetControl 300 series.

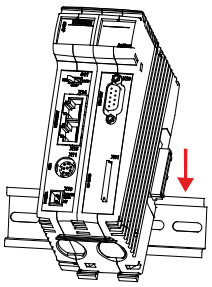
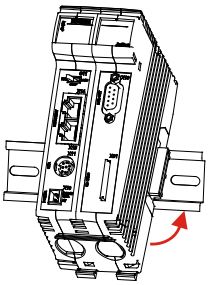
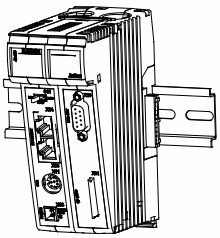
**Contents**


| Topic | Page |
|---|---|
| Installing the JC-360 on a DIN rail | 67 |
| Replacing the controller JC-360 | 68 |
| Removing the JC-360 from the DIN rail | 70 |

# Installing the JC-360 on a DIN rail

**Installing**

To install the controller on a rail to DIN EN 50022 proceed as follows:

| Step | Action | |
|---|---|---|
| 1 |  | Place the controller on the upper edge of the DIN rail. |
| 2 |  | Move the controller in the direction of the arrows until it snaps into place. |
| 3 |  | Installation of the controller on the DIN rail is now completed. |

**Related topics**

- **Replacing the controller JC-360** (see page 68)
- **Removing the controller JC-360 from the DIN rail** (see page 70)

# Replacing the controller JC-360

**Introduction**

At replacing a controller of the JetControl 300 series, the following configuration information is retained on the backplane module:

- IP address
- Subnet mask
- Gateway
- DNS server

**Switching between controller models**

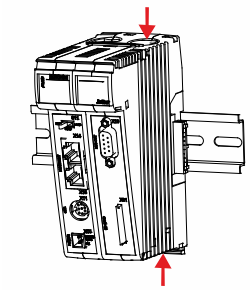
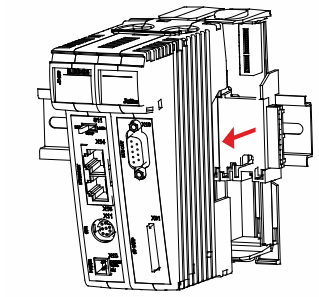
Any controller of the JetControl 300 series can be replaced by another controller model of this series. To increase performance, for example, the controller model JC-340-0 may be replaced by controller model JC-350-3.

Due to their compatibility, the following controllers are interchangeable:

- JC-340
- JC-350
- JC-360
- JC-360MC

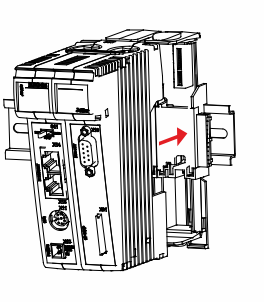
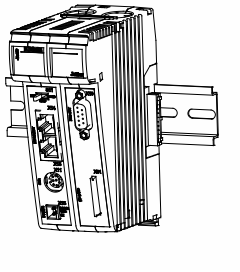
**Removing the controller**

To remove the JC-360 from the backplane module, proceed as follows:

| Step | Action | |
|---|---|---|
| 1 | Remove power from the JX3 station. | |
| 2 |  | Press the upper and lower latches located on the right half of the JC-360.<br>Keep the latches pressed. |
| 3 |  | Pull off the controller from the backplane module. |

**Installing the controller**

To install the JC-360 on the backplane module, proceed as follows:

| Step | Action | |
|---|---|---|
| 1 |  | Slide the controller onto the backplane module until the latches snap into place. |
| 2 |  | Installation of the controller to the backplane module is now completed. |

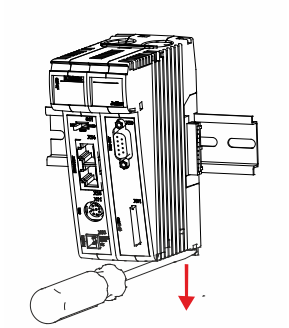
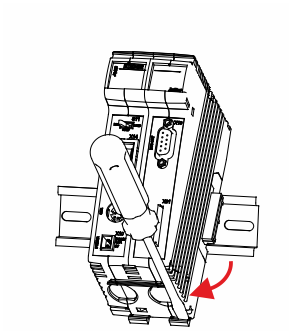
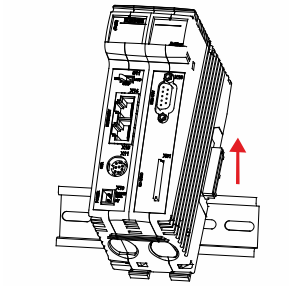
**Related topics**

- **Installing the JC-360 on a DIN rail** (see page 67)
- **Removing the JC-360 from the DIN rail** (see page 70)

## Removing the JC-360 from the DIN rail

**Removing**

To remove the JC-360 from a rail to DIN EN 50022 proceed as follows:

| Step | Action | |
|---|---|---|
| 1 | Remove power from the JX3 station. | |
| 2 | Slide the controller to the left. By doing so, the backplane module is disconnected from the JX3 peripheral modules. | |
| 3 |  | Pull down the right DIN rail latch using a flat-bladed screwdriver. |
| 4 |  | Swing the controller forward. |
| 5 |  | Remove the controller from the DIN rail. |

**Related topics**

- **Installing the JC-360 on a DIN rail** (see page 67)
- **Replacing the controller** (see page 68)

# 4.6 IP configuration

**Introduction**

This chapter describes the IP configuration for the controller JC-360. The following parameters can be set:

- IP address of the controller
- Subnet mask
- IP address of default gateway
- IP address of DNS server
- Controller name
- IP port number for the JetSym debugger
- Basic port number for communication via JetIP
- Name for AutoCopy command file

**Required skills of the network configurator**

To carry out IP configuration of the JC-360 knowledge of IP networks is required, such as

- IP addressing (IP address, port number, subnet masks etc.)
- FTP (connection setup, data transmission, etc.)

**Contents**


| Topic | Page |
|---|---|
| Factory settings | 72 |
| The configuration memory | 73 |
| The configuration file | 74 |
| Configuration registers | 78 |
| Changing the IP address of the controller | 79 |
| Setting the default IP address 192.168.10.15 | 80 |
| Setting the IP address via configuration file | 81 |
| Setting the IP address via configuration file and DIP switch | 82 |
| Setting the IP address via registers to be non-volatile | 84 |
| Setting the IP address during runtime | 86 |
| IP address in the GNN operating mode | 87 |
| Using names for IP addresses | 89 |

# Factory settings

**Introduction**

Before the JC-360 is shipped, various parameters are set to a certain value. The parameters can be changed by the user.

**Factory settings**

| Parameter | Value |
|---|---|
| IP address of the controller | 192.168.1.1 |
| Subnet mask | 255.255.255.0 |
| IP address of default gateway | 0.0.0.0 |
| IP address of DNS server | 0.0.0.0 |
| Controller name | JetControl360 |
| IP port number for debugger | 52000 |
| IP port number for JetIP | 50000 |
| Name for AutoCopy command file | /SD/autocopy.ini |
| DIP switch | DIP switch slider 1 = ON<br>All other DIP switch sliders = OFF |
| User's password *admin* | admin |
| User's password *system* | system |

# The configuration memory

**Introduction**

The controller reads the parameters for initializing the IP interface out of the configuration memory during the boot process. The user can access the data stored in the configuration memory in the following ways:

- A file located in the system directory of the file system lets you read out and modifiy data.
- Various registers or one registers let you read out data.

**Enabling conditions**

The controller reads out data located in the configuration memory only during the boot process. If you make changes to the configuration memory, reboot the controller for these changes to take effect. Only this way these changes take effect.

**Default values**

Before the controller further processes data from the configuration memory, it checks them for plausibility. If entries are invalid or absent, the controller uses the following default values:

| Parameter | Default value |
|---|---|
| IP address of the controller | 192.168.10.15 |
| Subnet mask | 255.255.255.0 |
| IP address of default gateway | 0.0.0.0 |
| IP address of DNS server | 0.0.0.0 |
| Controller name | JetControl360 |
| Suffix type of the name | 0 |
| IP port number for debugger | 52000 |
| IP port number for JetIP | 50000 |
| Name for AutoCopy command file | /SD/autocopy.ini |

**Storage location/controller replacement**

The configuration memory is located on the backplane module.

Owing to this approach, configuration data will be preserved when the function module is replaced.

**Related topics**

- **The configuration file** (see page 74)
- **Configuration registers** (see page 78)

# The configuration file

**Introduction**

The configuration file **config.ini** is used to access the configuration memory of the JC-360.

**Properties**

- You can access this file through the file system of the JC-360.
- For an FTP connection, the user must have administrator or system rights.
- This file is located in the folder **System**.
- You cannot delete the file, but only overwrite it.
- Formatting the Flash disk drive or the SD card leaves the file unchanged.

**File structure**

The configuration file is a text file the entries of which are grouped into several sections. The controller replaces missing IP configuration parameters by their default values.

**Configuration file - Example**

This is an example of a configuration file **config.ini**:

```
;<Productname> System Configuration
;Copyright (c) 2008 by Jetter AG, Ludwigsburg, Germany

[IP]
Address    = 192.168. 50.  1
SubnetMask = 255.255.255.  0
DefGateway = 192.168. 50. 11
DNSServer  = 192.168.  1. 44

[HOSTNAME]
SuffixType = 0
Name       = JetControl360

[PORTS]
JetIPBase = 50000
JVMDebug  = 52000

[FILES]
AutoCopyIni = /SD/autocopy.ini
```

**Section [IP]**

In section [IP] the required IP addresses and the subnet mask are specified.

| **Address** | |
|---|---|
| In the given example | 192.168.50.1 |
| Description | IP address of the JC-360<br>The DIP switch on the backplane module can overwrite the least significant byte. |
| Allowed values | ■ > 1.0.0.0<br>■ < 223.255.255.255 |
| Illegal values | ■ Network address<br>■ Broadcast address |
| In the event of an illegal value | JC-360 resets all four values to their defaults. |
| **SubnetMask** | |
| In the given example | 255.255.255.0 |
| Description | Sets the subnet mask |
| Allowed values | ■ >= 128.0.0.0 |
| Illegal values | ■ 1 and 0 mixed |
| In the event of an illegal value | JC-360 resets all four values to their defaults. |
| **DefGateWay** | |
| In the given example | 192.168.50.11 |
| Description | IP address of the gateway to other subnets;<br>The JC-360 must be able to reach the subnet (Address/SubnetMask), otherwise it will set this parameter to 0.0.0.0. |
| Allowed values | ■ >= 0.0.0.0 and<br>■ < 223.255.255.255 |
| Illegal values | ■ Network address<br>■ Broadcast address<br>■ A value (Address/SubnetMask) which cannot be reached by the JC-360.<br>■ The "Address" value |
| In the event of an illegal value | JC-360 sets the value to 0.0.0.0 |
| **DNSServer** | |
| In the given example | 192.168.1.44 |
| Description | IP address of the server for the Domain Name System |
| Allowed values | ■ >= 0.0.0.0 and<br>■ < 223.255.255.255 |
| In the event of an illegal value | JC-360 sets the value to 0.0.0.0 |

**Section [HOSTNAME]**

In section [HOSTNAME] the name of the JC-360 is specified. If desired, the JC-360 automatically generates an individual name. The JC-360 presently does not use the host name.

| **SuffixType** | |
|---|---|
| In the given example | 0 |
| Description | The type of the automatically generated suffix is attached to the controller name |
| Allowed values | ■ 0: No suffix<br>■ 1: Low-order byte of the IP address in decimal notation<br>■ 2: Low-order byte of the IP address in hexadecimal notation |
| In the event of an illegal value | 0 |
| **Name** | |
| In the given example | JetControl360 |
| Description | Specifies the name of the JC-360 |
| Allowed values | ■ First character: 'A' ... 'Z', 'a' ... 'z'<br>■ Next characters: 'A' ... 'Z', 'a' ... 'z', '0' ... '9', '-' |
| In the event of an illegal value | JetControl360 |

**Section [PORTS]**

In section [PORTS] the IP port numbers of data and debug servers within the JC-360 are specified. The IP port numbers must be consistent with, for example, the port numbers set in JetSym.

| **JetIPBase** | |
|---|---|
| In the given example | 50000 |
| Description | IP port for OS update and communication between controllers |
| Allowed values | ■ 1024 ... 65535 |
| In the event of an illegal value | 50000 |
| **JVMDebug** | |
| In the given example | 52000 |
| Description | IP port for debugger/setup in JetSym |
| Allowed values | ■ 1024 ... 65535 |
| In the event of an illegal value | 52000 |

**Section [FILES]**

In the [FILES] section, the name of the command file for the AutoCopy function has been entered.

| **AutoCopyIni** | |
|---|---|
| In the given example | /SD/autocopy.ini |
| Function | Command file for the AutoCopy function |
| Allowed values | Valid name of path and file |
| In the event of an illegal value | /SD/autocopy.ini |

**Changing the IP configuration**

| Step | Action |
|---|---|
| **1** | Create on your PC a configuration file named **config.ini** using a text editor and make the corresponding entries. |
| **2** | Open an FTP connection between the PC and the JC-360. |
| **3** | Log in as user with administrator or system rights.<br>Default login information:<br>User: admin, Password: admin<br>User: system; Password: system |
| **4** | Browse to directory */System* of the JC-360. |
| **5** | Copy the configuration file named **config.ini**, which has been created by you, to the JC-360. |
| **6** | Clear the FTP connection. |
| **7** | Reboot the JC-360.<br>**Result**:<br>The new configuration is active. |

As an alternative, you can change the IP configuration via the configuration registers.

**Related topics**

- **The configuration memory** (see page 73)
- **Configuration registers** (see page 78)

# Configuration registers

**Introduction**

Configuration registers let you read the parameters of the IP configuration. A range of registers holds the data contained in the configuration memory. Another range contains the parameters used for initializing the IP interface.

**Register numbers**

The basic register numbers of both ranges are dependent on the device. The register number is calculated by adding the number of the module register (MR) to the number of the basic register.

| Device | Data range | Basic register number | Register numbers |
|---|---|---|---|
| JC-360 | Configuration memory | 101100 | 101100 ... 101165 |
| | Parameters used | 101200 | 101200 ... 101265 |

**Configuration registers**

The following table lists the registers of both ranges, as well as their connection to the entries in the configuration file **/System/config.ini**:

| Register | Section in config.ini | Name in config.ini | Description |
|---|---|---|---|
| **MR 0** | IP | `Address` | IP address of the controller |
| **MR 1** | | `SubnetMask` | Sets the subnet mask |
| **MR 2** | | `DefGateWay` | IP address of the gateway to other subnets |
| **MR 3** | | `DNSServer` | IP address of the server for the Domain Name System |
| **MR 32** | HOSTNAME | `SuffixType` | The type of the automatically generated suffix is attached to the controller name |
| **MR 33 through 51** | | `Name` | Specifies the controller name |
| **MR 64** | PORTS | `JetIPBase` | IP port number for OS update and communication between controllers |
| **MR 65** | | `JVMDebug` | IP port number for debugger/setup in JetSym |
| **MR 80** | FILES | `AutoCopyIni` | Name for AutoCopy command file |

**Related topics**

- **The configuration memory** (see page 73)
- **The configuration file** (see page 74)

# Changing the IP address of the controller

**Introduction**

To be able to communicate with the controller JC-360 via Ethernet, you must set an unambiguous IP address on the controller.

**Replacing the controller**

The IP address is stored to the backplane module. When you install another controller of the JetControl 300 series to the backplane module, the following configuration data are preserved:

- IP address of the controller
- Subnet mask
- IP address of default gateway
- IP address of DNS server
- Controller name
- Suffix type of the name
- IP port number for debugger
- IP port number for JetIP
- Name for AutoCopy command file

**Configuration options**

You can configure the IP address in the following ways:

- Default IP address
- Configuration via file **config.ini**
- Configuration via file **config.ini** and DIP switch
- Configuration via the configuration registers
- Configuration during runtime via special registers

**Changing the IP address**

| Step | Action |
|---|---|
| **1** | Remove power from the controller JC-360. |
| **2** | Remove the controller enclosure from the backplane module. |
| **3** | Make the corresponding DIP switch settings. |
| **4** | Reinstall the enclosure on the backplane module. |
| ⇨ | Following restart, the controller JC-360 can be reached at the new IP address. |

**Related topics**

- **Replacing the controller** (see page 68)
- **Default IP address 192.168.10.15** (see page 80)
- **Setting the IP address via configuration file** (see page 81)
- **Setting the IP address via configuration file and DIP switch** (see page 82)
- **Setting the IP address during runtime** (see page 86)

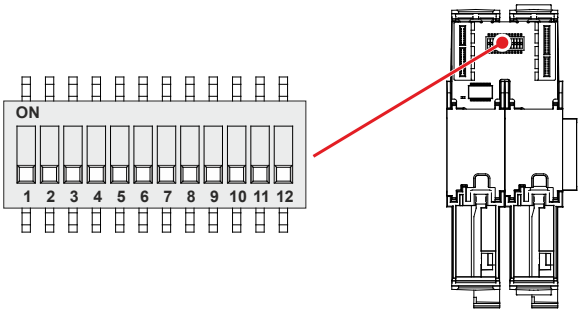
## Setting the default IP address 192.168.10.15

**Default IP address**

The controller JC-360 has got default IP address 192.168.10.15. You may change the IP address of the JC-360 to its default IP address at any time.

**DIP switch settings**

To set the module to its default IP address 192.168.10.15, move the DIP switch sliders to the positions shown below:



# Setting the IP address via configuration file

**Setting the IP address**

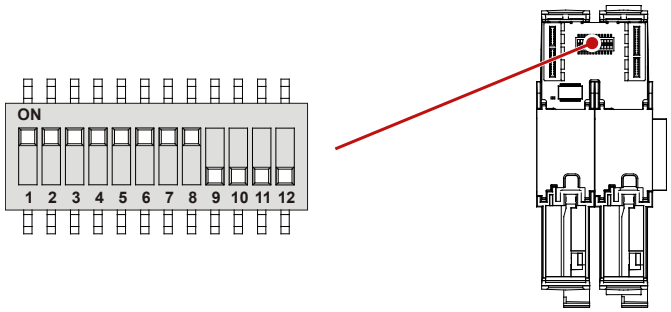
You can set the IP address of the controller JC-360 in the **config.ini** file.

```
[IP]
Address     = aaa.bbb.ccc.ddd
...
```

| Element | Description |
|---|---|
| `Address` | Enter the IP-address into this line. |
| `aaa` | First byte of IP address |
| `bbb` | Second byte of IP address |
| `ccc` | Third byte of IP address |
| `ddd` | Fourth byte of IP address |

**DIP switch settings**

The following DIP switch settings cause the controller JC-360 to read out the IP address from the file **config.ini**:




**Transferring the configuration file**

| Step | Action |
|---|---|
| **1** | Establish an FTP connection to the JC-360. |
| **2** | Log in as user with administrator or system rights.<br>Default login information:<br>User: *admin*; Password: *admin*<br>User: *system*; Password: *system* |
| **3** | Open the folder **System**. |
| **4** | Copy the file **config.ini** into the folder **System**. |
| **5** | Clear the FTP connection. |
| **6** | Reboot the JC-360. |

## Setting the IP address via configuration file and DIP switch

**Introduction**

You can set the IP address of the JC-360 via the configuration file **config.ini** and the DIP switch sliders. To this end, set the three upper bytes of the IP address in the**config.ini** file, and the lower byte using the DIP switch sliders.

**Configuration file - Setting the IP address**

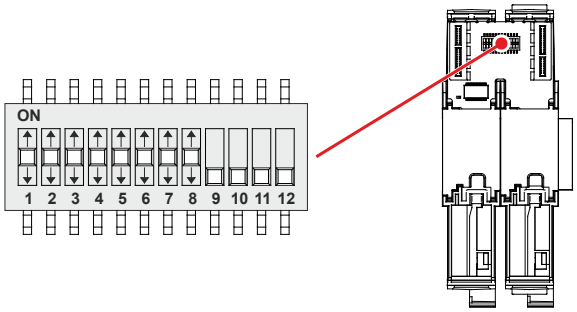
Set the upper three bytes of the IP address of the JC-360 in the configuration file **config.ini** file.

```
[IP]
Address     = aaa.bbb.ccc.1
...
```

| Element | Description |
|---|---|
| `Address` | Line for entering the upper 3 bytes of the IP address |
| `aaa` | First byte of IP address |
| `bbb` | Second byte of IP address |
| `ccc` | Third byte of IP address |
| `1` | Dummy entry, must have got value one. |

**DIP switch settings**

The following DIP switch settings cause the JC-360 to read out the IP address from the file **config.ini** and the DIP switches:




| DIP switch | | | | | | | | IP address |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | |
| OFF | OFF | OFF | OFF | OFF | OFF | OFF | OFF | Default IP address |
| **ON** | OFF | OFF | OFF | OFF | OFF | OFF | OFF | aaa.bbb.ccc.1 |
| OFF | **ON** | OFF | OFF | OFF | OFF | OFF | OFF | aaa.bbb.ccc.2 |
| **ON** | **ON** | OFF | OFF | OFF | OFF | OFF | OFF | aaa.bbb.ccc.3 |
| ... | | | | | | | | |
| OFF | **ON** | **ON** | **ON** | **ON** | **ON** | **ON** | **ON** | aaa.bbb.ccc.254 |
| **ON** | **ON** | **ON** | **ON** | **ON** | **ON** | **ON** | **ON** | from config.ini |

**Transferring the configuration file**

| Step | Action |
|---|---|
| **1** | Establish an FTP connection to the JC-360. |
| **2** | Log in as user with administrator or system rights.<br>Default login information:<br>User: *admin*; Password: *admin*<br>User: *system*; Password: *system* |
| **3** | Open the folder **System**. |
| **4** | Copy the file **config.ini** into the folder **System**. |
| **5** | Clear the FTP connection. |
| **6** | Reboot the JC-360. |

## Setting the IP address via registers to be non-volatile

**Introduction**

The IP interface is initialized by the settings in the configuration memory during the boot process.

The following non-volatile settings can also be changed via register:

- IP address of the controller
- Subnet mask
- IP address of default gateway
- IP address of DNS server
- Host name and suffix type
- Port numbers for JetIP and the JetSym debugger
- Name for AutoCopy command file

**Registers - Overview**

Overview over the configuration memory registers:

| Registers | Description |
|---|---|
| **101200** | IP address |
| **101201** | Subnet mask |
| **101202** | IP address of default gateway |
| **101203** | IP address of DNS server |
| **101232** | Host name suffix type |
| **101233 through 101251** | Host name |
| **101264** | Port number for JetIP |
| **101265** | Port number for STX debugger |
| **101280 through 101298** | Name for AutoCopy command file |
| **101299** | Saving the settings (0x77566152) |

**Setting the configuration values to be non-volatile**

To change the configuration values to become non-volatile, proceed as follows:

| Step | Action |
|---|---|
| **1** | Enter the desired configuration into one or several registers within the range from 101200 to 101298. |
| **2** | To have the controller take over the values, you must enter a password.<br>For this, write value 2002149714 (0x77566152) to register 101299. |
| **3** | Wait for the controller to write value 0 into MR 101299.<br>The save process is now completed. |
| **4** | Boot the controller. |

**Result:** The settings are completed. Communication is possible again.

**Important note:**
The EEPROM data on the backplane module allow for 100,000 write cycles.
We urgently recommend the following workflow:
First read out the value, then compare it. Only if the value has changed, start writing.

**Effect**

Write to register 101299 to have the controller take the following steps:

- The controller creates a configuration file out of the values.
- It saves the configuration file to the backplane module as **/System/config.ini**.
- If you have entered comments and formatting details into this file, the comments and formatting details will get lost during this process.

**Related topics**

- **The configuration memory** (see page 73)
- **Setting the IP address during runtime** (see page 86)
- **Setting the IP address via configuration file** (see page 81)
- **Setting the IP address via configuration file and DIP switch** (see page 82)

## Setting the IP address during runtime

**Introduction**

The IP interface is initialized by the settings in the configuration memory during the boot process.

The following settings can also be changed via registers to be non-remanent:

- IP address of the controller
- Subnet mask
- IP address of default gateway

**Important note**

The settings made during runtime do not change the parameters in the configuration memory. At de-energizing the controller, your settings will be lost.

**Prerequisites**

- While settings are being made, no communication via IP interface is allowed. Otherwise, this would lead to a loss of data.
- The values entered must be valid. This can be ensured, e.g. by including a validity check in the application program.
  This is important because there is no check if you set the parameters during runtime of the controller.

**Register overview**

| Register | Description |
|---|---|
| **104531** | IP address of the JC-360 |
| **104532** | Subnet mask |
| **104533** | IP address of default gateway |

**Setting IP addresses and subnet mask**

To set the IP address and the subnet mask, proceed as follows:

| Step | Action |
|---|---|
| **1** | Enter the value 0.0.0.0 into 104533. |
| **2** | Enter the value 0.0.0.0 into 104532. |
| **3** | Enter the desired IP address into register 104531. |
| **4** | Enter the desired subnet mask into register 104532. |
| **5** | Enter the desired IP address of the Default Gateway into 104533. |

**Result:**

The settings are completed. Communication is possible again.

**Related topics**

- **The configuration memory** (see page 73)

# IP address in the GNN operating mode

**Introduction**

In GNN mode (Global Network Number mode), the JC-360 functions as a network node within a greater controller network. It derives its IP address at booting from the configuration file **config.ini**. Next, the main controller, which is the NetConsistency master, commands the network node to log in with the GNN from the NetConsistency master. After accepting the network node, the main controller compares the set IP configuration with the actual IP configuration of the network node. If a difference results, the main controller makes the corresponding changes in the set IP configuration in the network node.

**Configuration file - Setting the IP address**

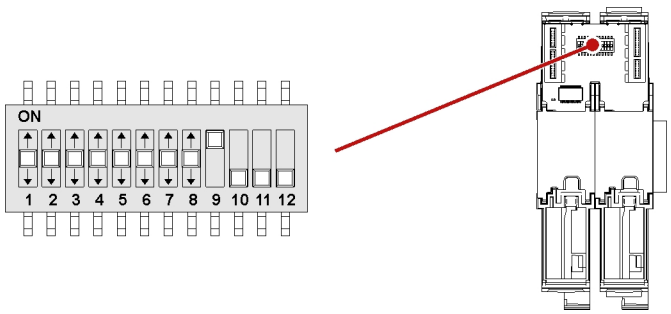
Set the IP address in the configuration file **config.ini**.

```
[IP]
Address     = aaa.bbb.ccc.ddd
...
```

| Element | Description |
|---|---|
| `Address` | Line for entering the IP address |
| `aaa` | First byte of IP address |
| `bbb` | Second byte of IP address |
| `ccc` | Third byte of IP address |
| `ddd` | Fourth byte of IP address |

**DIP switch settings**

The following DIP switch settings cause the JC-360 to read out the IP address from the file **config.ini** and the GNN out of the lower eight positions of the DIP switches:




Valid values for the GNN: 1 ... 199.

The DIP switches are binary-coded.

| DIP switch | | | | | | | | GNN |
|---|---|---|---|---|---|---|---|---|
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | |
| OFF | OFF | OFF | OFF | OFF | OFF | OFF | OFF | Invalid |
| **ON** | OFF | OFF | OFF | OFF | OFF | OFF | OFF | 1 |
| OFF | **ON** | OFF | OFF | OFF | OFF | OFF | OFF | 2 |

| DIP switch | | | | | | | | GNN |
|---|---|---|---|---|---|---|---|---|
| **ON** | **ON** | OFF | OFF | OFF | OFF | OFF | OFF | 3 |
| ... | | | | | | | | |
| **ON** | **ON** | **ON** | OFF | OFF | OFF | **ON** | **ON** | 199 |
| ... | | | | | | | | Invalid |
| **ON** | **ON** | **ON** | **ON** | **ON** | **ON** | **ON** | **ON** | Invalid |

# Using names for IP addresses

**Introduction**

Names can be specified as IP addresses for target systems, e.g. when configuring the e-mail client. The JC-360 resolves the names into IP addresses. A configuration file or the Domain Name System is used to assign names to their corresponding IP address.

**Name resolution**

Names are resolved to IP addresses in the following way:

| Step | Description | |
|---|---|---|
| **1** | During the boot process the JC-360 reads the IP address of the DNS server out of the configuration memory. | |
| **2** | During the boot process the JC-360 reads the file **/etc/hosts**. The JC-360 creates a translation table with the names and IP addresses found in this file. | |
| **3** | After the boot process the JC-360 detects a name instead of an IP address. | |
| **4** | Based on this translation table, the JC-360 tries to resolve the name into a related IP address. | |
| | **If ...** | **... then ...** |
| | ... the JC-360 has resolved the name, ... | ... proceed with step 6. |
| | ... the JC-360 has not resolved the name, ... | ... proceed with step 5. |
| **5** | The JC-360 tries to resolve the name into a related IP address by sending a request to the DNS server. | |
| | **If ...** | **... then ...** |
| | ... the JC-360 has resolved the name, ... | ... it enters the name and IP address into the translation table;<br>--> proceed with step 6. |
| | ... the JC-360 has not resolved the name, ... | ... the JC-360 aborts the function, e.g. the system function for sending an e-mail, and issues an error message. |
| **6** | The JC-360 uses the IP address resolved for further communication. | |

**Configuration file**

This configuration file **hosts** holds the static assignment between names and IP addresses. The JC-360 reads this file once during boot-up.

File format: Text
Location: /etc
File name: hosts

**Example**

```
# Example - hosts file for JC-3xx
192.168.33.209    jetter_mail
192.168.33.208    jetter_demo
192.168.1.1       JC340
192.168.1.2       JC350
```

**Domain Name System (DNS)**

If a name cannot be found in the file **/etc/hosts**, the JC-360 tries to obtain the corresponding IP address from a DNS server. During boot-up, the JC-360 reads the IP address of the DNS server out of the configuration memory.

**Related topics**

- **The configuration memory** (see page 73)

# 4.7 Engineering of JX3 station equipped with a JC-360

**Introduction**

This chapter describes how to engineer a JX3 station equipped with a JC-360.

**JX3 station**

A JX3 station consists of a JX3-BN-xxx bus node or a JC-3xx controller and JX3 peripheral modules connected to it.

**Number of connectable JX3 modules**

The possible number of JX3 modules depends on the following parameters:

- Maximum number of JX3 modules
- Maximum data exchange rate
- Maximum power consumption

**Contents**


| Topic | Page |
|---|---|
| Limitations to be taken into account when engineering a JX3 station | 92 |

# 4.7.1 Limitations to be taken into account when engineering a JX3 station

**Introduction**

This chapter covers the following limitations of a JX3 station:

- Maximum number of JX3 modules
- Maximum data interchange rate
- Maximum power consumption

Take all three limitations into account when engineering a JX3 station and match the JX3 peripheral modules considering all limiting factors. If you need more JX3 peripheral modules for your plant, you can expand it by means of JX3-BN-xxx.

**JX3 system bus configurator**

The JX3 system bus configurator assists you in detecting limitations through engineering a real JX3 station.

In this case, enter the amount of your JX3 modules into an Excel file. The JX3 system bus configurator calculates the possibility of the above named limitations.

There are two Excel sheets **JX3 systembus data** and **JX3 systembus power**.

**JX3 systembus data**

Here, the maximum number of JX3 modules and maximum data interchange rate is determined.

| configuration of JX3-station (data) | | |
|---|---|---|
| total number of modules | | |
| In-size of JX3-station | | |
| Out-size of JX3-station | | |
| **module name** | **description** | **number** |
| JX3-AI4 | Analog Input Module | |
| JX3-AO4 | Analog Output Module | 11 |
| JX3-CNT | Counter Module | |
| JX3-DI16 | Digital Input Module | |
| JX3-DIO16 | Digital Input/Output Module | |
| JX3-DMS2 | Strain Gage Module | 0 |
| JX3-DO16 | Digital Output Module | 3 |
| JX3-MIX1 | Multi-Purpose Expansion Module | 0 |
| JX3-MIX2 | Multi-Purpose Expansion Module | |
| JX3-THI2-RTD | Pt100/Pt1000 Temperature Measurement | |
| JX3-THI2-TC | Thermocouple Temperature Measurement | |
| | | |
| | | |

The following fields marked red convey the following meaning:

- **Total number of modules**
  The maximum number of 16 JX3 modules per JX3 station has been exceeded.
- **IN size of JX3 station**
  The aggregated input factor has been exceeded.

- **OUT size of JX3 station**
  The aggregated output factor has been exceeded.

**JX3 systembus power**

Here, the maximum number of JX3 modules and maximum power consumption is determined.

| configuration of JX3-station (power) | | |
|---|---|---|
| number of Modules | | |
| JC-3xx/JX3-BN-ETH | | |
| JX3-BN-CAN | | |
| JX3-PS1 | | |
| **module name** | **description** | **number** |
| JX3-AI4 | Analog Input Module | |
| JX3-AO4 | Analog Output Module | |
| JX3-AO4 Current | Analog Output Module (in Current Mode) | 6 |
| JX3-CNT | Counter Module | |
| JX3-DI16 | Digital Input Module | |
| JX3-DIO16 | Digital Input/Output Module | 0 |
| JX3-DMS2 | Strain Gage Module | |
| JX3-DO16 | Digital Output Module | 0 |
| JX3-MIX1 | Multi-Purpose Expansion Module | |
| JX3-MIX2 | Multi-Purpose Expansion Module | |
| JX3-THI2-RTD | Pt100/Pt1000 Temperature Measurement | |
| JX3-THI2-TC | Thermocouple Temperature Measurement | |
| | | |
| | | |

The following fields marked red convey the following meaning:

- **Number of modules**
  The maximum number of 8 JX3 modules connected to JC-3xx, JX3-BN-xxx, or JX3-PS1 has been exceeded.
  Integrate the JX3-PS1 in your JX3 station.
- **JC-3xx/JX3-BN-ETH**
  The power consumption of the JX3 modules which are directly connected to a JC-3xx or a JX3-BN-ETH is too high.
  Insert a JX3-PS1 into your JX3 station.
- **JX3-BN-CAN**
  The power consumption of the JX3 modules which are directly connected to a JX3-BN-CAN is too high.
  Insert a JX3-PS1 into your JX3 station.
- **JX3-PS1**
  The power consumption of the JX3 modules which are directly connected to a power supply module JX3-PS1 is too high.
  Insert a JX3-PS1 into your JX3 station.

**JX3 system bus configurator - Download**

Jetter AG provide the JX3 system bus configurator on their **homepage http://www.jetter.de**. You can find the JX3 system bus configurator for download at *Industrial automation - support - downloads - 08_miscellaneous - jx3_system_bus - tools*.

**Contents**


| Topic | Page |
|---|---|
| Limitations of the maximum number of modules | 95 |
| Limitations of the modules' data exchange rates | 96 |
| Limitation depending on the power consumption of the modules | 99 |

# Limitations of the maximum number of modules

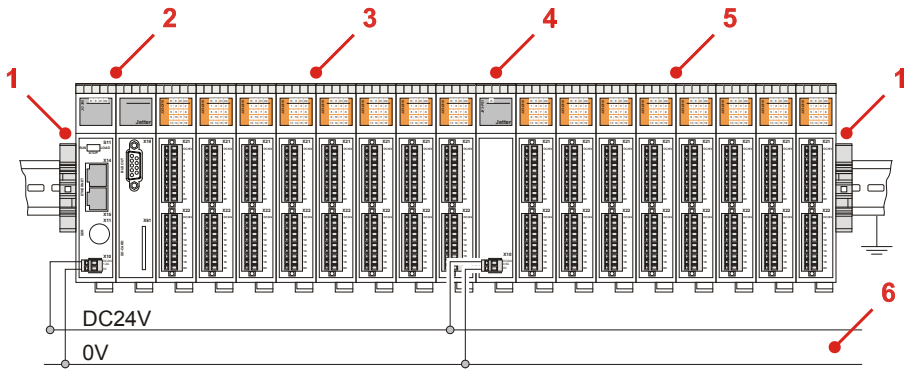
**Limited maximum number**

The maximum number of JX3 peripheral modules per JX3 station is limited. The following rule applies:

- Up to 16 JX3 peripheral modules can be integrated into a JX3 station.
- You can directly connect up to eight JX3 peripheral modules to a JC-3xx controller.
- Downstream the eighth JX3 peripheral module you must insert a JX3-PS1 power supply module.
- In addition, you can connect up to eight JX3 peripheral modules to a JX3-PS1 power supply module.

**Fully equipped JX3 station**

The following illustration shows a JX3 station with a JC-3xx controller and 16 JX3 peripheral modules. Downstream the eighth JX3 peripheral module a JX3-PS1 power supply module has been inserted.




| Number | Part | Description |
|---|---|---|
| **1** | End clamp for the DIN rail | For securing JX3 modules on the DIN rail |
| **2** | JC-3xx | Controller and power supply for the first eight JX3 peripheral modules |
| **3** | JX3 peripheral modules | Eight JX3 peripheral modules |
| **4** | JX3-PS1 | Power supply module for the next eight JX3 peripheral modules |
| **5** | JX3 peripheral modules | Eight JX3 peripheral modules |
| **6** | Power supply lines DC 24 V and 0 V | Logic voltage supply for all modules of the JX3 station |

## Limitations of the modules' data exchange rates

**Introduction**

There are two options for connecting remote peripheral modules: via JX3-BN-CAN using the JX2 system bus protocol, or via JX3-BN-ETH using the Ethernet system bus protocol. JX3 modules of a JX3 station exchange data either with a JC-3xx controller, or with bus node modules JX3-BN-CAN and JX3-BN-ETH. The maximum number of input and output data and of JX3 peripheral modules in a JX3 system is limited.

**Evaluation of input and output factors**

The following table lists the factors for input and output data.

- Add the input factor of all modules used and calculate the aggregated input factor.
- Add the output factor of all modules used and calculate the aggregated output factor.
- Compare the aggregated factor for inputs/outputs with the allowed maximum factor:

$$\sum Eingangsdaten_{max} = \sum n_{Peripheriemodule} \cdot Faktor_{jeModul} \leq 88$$
$$\sum Ausgangsdaten_{max} = \sum n_{Peripheriemodule} \cdot Faktor_{jeModul} \leq 88$$

where n <= 16.

| Peripheral module | Input factor per module | Output factor per module |
|---|---|---|
| JX3-AI4 | 10 | 0 |
| JX3-AO4 | 2 | 8 |
| JX3-CNT | 10 | 0 |
| JX3-DI16 | 4 | 0 |
| JX3-DIO16 | 4 | 2 |
| JX3-DMS2 | 10 | 0 |
| JX3-DO16 | 2 | 2 |
| JX3-MIX1 | 16 | 6 |
| JX3-MIX2 | 16 | 6 |
| JX3-THI2-RTD | 10 | 0 |
| JX3-THI2-TC | 10 | 0 |

**Devices not to be taken into account**

The following devices are not taken into account when calculating the aggregated input/output factor:

- Controller JC-3xx
- Bus node JX3-BN-ETH
- Bus node JX3-BN-CAN
- Power supply module JX3-PS1

**JX3 station - Maximum configuration**

The table below lists the maximum allowed number of modules and the aggregated factor of input data and output data.

| Maximum number of peripheral modules | Aggregated input factor | Aggregated output factor |
|---|---|---|
| 16 | 88 | 88 |

**Engineering steps**

To engineer a JX3 station, proceed as follows:

| Step | Action | |
|---|---|---|
| **1** | Do not exceed the maximum number of 16 modules per JX3 station. | |
| **2** | Calculate the aggregated input factor by adding the input factor per module.<br>**Example:**<br>2 JX3-DI16 and 8 JX3-AI4 modules are connected to a controller JC-3xx. 2 JX3-DI16 with input factor 4 makes 8; 8 JX3-AI4 with input factor 10 makes 80 => aggregated factor is 88. | |
| **3** | Calculate the aggregated output factor by adding the output factor per module.<br>**Example:**<br>2 JX3-DI16 and 8 JX3-AI4 modules are connected to a controller JC-3xx. 2 JX3-DI16 with input factor 0 makes 0; 8 JX3-AI4 with input factor 0 makes 0 => aggregated factor is 0. | |
| **4** | **If ...** | **... then ...** |
| | ... the aggregated input factor makes 88, ... | ... the JX3 station is fully equipped. |
| | ... the aggregated output factor makes 88, ... | ... the JX3 station is fully equipped. |
| | ... the number of peripheral modules is 16, ... | ... the JX3 station is fully equipped. |

If one of the three values has been exceeded, a new JX3 station must be added. For this, apply a bus node, e.g. a JX3-BN-ETH.

**Example**

You want to connect 11 JX3-AO4 and 5 JX3-THI2-RTD to a controller JC-3xx.

**Input factors in our example**

| Quantity | Module | Input factor per module | Aggregated factor |
|---|---|---|---|
| **11** | JX3-AO4 | 2 | 22 |
| **5** | JX3-THI2-RTD | 10 | 50 |

| | |
|---|---|
| **Aggregated input factor** | **72** |

**Output factors in our example**

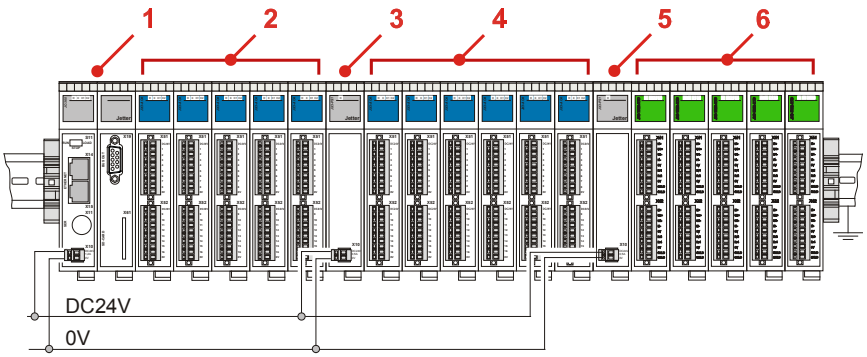
| Quantity | Module | Output factor per module | Aggregated factor |
|---|---|---|---|
| **11** | JX3-AO4 | 8 | 88 |
| **5** | JX3-THI2-RTD | 0 | 0 |

| | |
|---|---|
| **Aggregated output factor** | **88** |

**Conclusion of I/O factor calculation**

You can engineer the JX3 station out of 11 JX3-AO4 and 5 JX3-THI2-RTD as described below. One JX3 station will be sufficient.

Depending on the view on the power consumption unit, you must mount a JX3-PS1 after a certain number of JX3 modules as is shown in the illustration.




| Number | Part | Description |
|---|---|---|
| **1** | JC-3xx | Controller |
| **2** | JX3-AO4 | Modules 1 to 5 supplied by JC-3xx (1) |
| **3** | JX3-PS1 | Power supply module |
| **4** | JX3-AO4 | Modules 6 to 11 supplied by JX3-PS1 (3) |
| **5** | JX3-PS1 | Power supply module |
| **6** | JX3-THI2-RTD | Modules 12 to 16 supplied by JX3-PS1 (5) |

## Limitation depending on the power consumption of the modules

**Introduction**

JX3 modules of a JX3 station are supplied with logic voltage either by a JC-3xx controller, or a JX3-BN-xxx bus node, or a JX3-PS1 power supply module. Each of these modules is able to supply up to 8 downstream JX3 modules with logic voltage.

Certain JX3 modules, however, have a higher power consumption which may reduce the number of JX3 modules that can be connected to one JX3 station. Depending on the power consumption, you must equip the JX3 station with additional power supply modules JX3-PS1.

**Allowed power consumption**

The following table shows the allowed power consumption of JX3 modules connected to the right (downstream) of the power supply module.

| Power Supply Module | Power consumption $P_{24\ V}$ | Power consumption $P_{5\ V}$ |
|---|---|---|
| JC-3xx | 18 W | 6 W |
| JX3-BN-ETH | 18 W | 6 W |
| JX3-BN-CAN | 22 W | 6 W |
| JX3-PS1 | 24 W | 6 W |

**Devices not to be taken into account**

The following devices are not taken into account when calculating power consumption:

- Controller JC-3xx
- Bus node JX3-BN-ETH
- Bus node JX3-BN-CAN
- Power supply module JX3-PS1

**Engineering steps**

To engineer a JX3 station, proceed as follows:

| Step | Action | |
|---|---|---|
| 1 | Gather the following information from the technical data contained in the manual of your JX3 module:<br>■ Current consumption from the logic voltage of the JX3 system bus: $I_{5\,V}$<br>■ Current consumption from the additional voltage of the JX3 system bus: $I_{24\,V}$ | |
| 2 | Calculate the power consumption of the JX3 modules:<br>$P_{5V} = 5V \cdot I_{5V}$<br>$P_{24V} = 24V \cdot I_{24V} + \frac{P_{5V}}{0.85}$ | |
| 3 | Add the power consumption of JX3 modules included in the JX3-station. Start with the first JX3 module connected to the JC-3xx controller, or to the JX3-BN-xxx bus node. | |
| 4 | Check whether the allowed power consumption has been exceeded. | |
| 5 | **If ...** | **... then ...** |
| | ... the allowed power consumption $P_{5\,V}$ has been reached, ... | ... insert a JX3-PS1 power supply module upstream the next JX3 module. |
| | ... the allowed power consumption $P_{24V}$ has been reached, ... | ... insert a JX3-PS1 power supply module upstream the next JX3 module. |
| | ... 8 JX3 modules have been connected ... | ... insert a JX3-PS1 power supply module upstream the next JX3 module. |
| | ... 16 JX3 modules have been connected ... | ... the JX3 station is fully equipped. |

**Example**

You want to connect 11 JX3-AO4 and 5 JX3-THI2-RTD to a controller JC-3xx. Check by taking the following steps how the JX3 station must be engineered to keep power consumption from exceeding the permitted value.

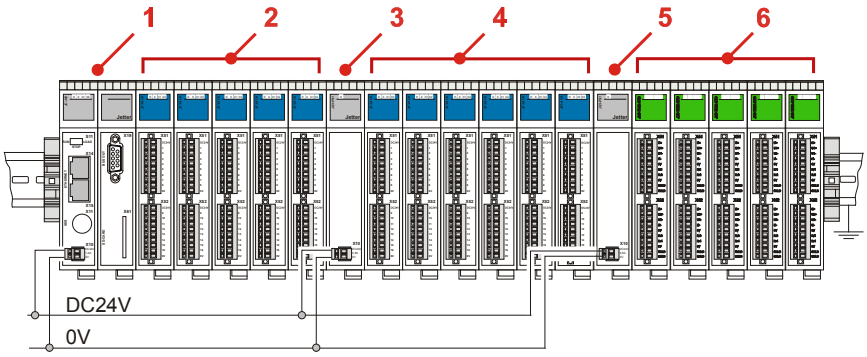
| Step | Action |
|---|---|
| 1 | Determine the current consumption of the JX3 module JX3-AO4 given in the technical data:<br>■ Current consumption from the logic voltage of the JX3 system bus: 70 mA<br>■ Current consumption from the additional voltage of the JX3 system bus: 120 mA |
| 2 | Calculate the power consumption of an JX3-AO4:<br>$P_{5V} = 5V \cdot 70mA = 0.35W$<br>$P_{24V} = 24V \cdot 120mA + \frac{0.35W}{0.85} = 3.29W$ |
| 3 | Add the power consumption of n JX3-AO4:<br>$\sum P_{5V} = n \cdot P_{5V}$<br>$\sum P_{24V} = n \cdot P_{24V}$ |

<table>
<tr><th>Step</th><th colspan="2">Action</th></tr>
<tr><td>⇨</td><td colspan="2">For 11 JX3-AO4 modules, the aggregated power consumption is as follows:<br>$\sum P_{5V} = 11 \cdot P_{5V} = 3.85W$<br>$\sum P_{24V} = 11 \cdot P_{24V} = 36.2W$</td></tr>
<tr><td rowspan="3">4</td><td colspan="2">Check for permitted power consumption.</td></tr>
<tr><td>For JC-3xx:</td><td>For JX3-PS1:</td></tr>
<tr><td>$\sum P_{5V} \leq 6W$<br>$\sum P_{24V} \leq 18W$</td><td>$\sum P_{5V} \leq 6W$<br>$\sum P_{24V} \leq 24W$</td></tr>
<tr><td>⇨</td><td colspan="2">The number of JX3-AO4 modules is limited by the power consumption at 24 V:<br>For JC-3xx:<br>$n = \frac{\sum P_{24V}}{P_{24V}} = \frac{18W}{3.29W} = 5.47 \Rightarrow 5$<br>For JX3-PS1:<br>$n = \frac{\sum P_{24V}}{P_{24V}} = \frac{24W}{3.29W} = 7.29 \Rightarrow 7$</td></tr>
<tr><td>5</td><td colspan="2">This module lets you directly connect up to 5 JX3-AO4 to a JC-3xx. Then, insert a JX3-PS1. Then, you can connect the remaining 6 JX3-AO4 to this JX3-PS1.</td></tr>
<tr><td>6</td><td colspan="2">Determine the current consumption of the JX3 module JX3-THI2-RTD given in the technical data:<br>■ Current consumption from the logic voltage of the JX3 system bus: 210 mA<br>■ Current consumption from the additional voltage of the JX3 system bus: 0 mA</td></tr>
<tr><td>7</td><td colspan="2">Calculate the power consumption of an JX3-THI2-RTD:<br>$P_{5V} = 5V \cdot 210mA = 1.05W$<br>$P_{24V} = 24V \cdot 0mA + \frac{1.05W}{0.85} = 1.24W$</td></tr>
<tr><td>8</td><td colspan="2">Add the power consumption of n JX3-THI2-RTD:<br>$\sum P_{5V} = n \cdot P_{5V}$<br>$\sum P_{24V} = n \cdot P_{24V}$</td></tr>
<tr><td>⇨</td><td colspan="2">For 5 JX3-THI2-RTD modules, the aggregated power consumption is as follows:<br>$\sum P_{5V} = 5 \cdot P_{5V} = 5.25W$<br>$\sum P_{24V} = 5 \cdot P_{24V} = 6.2W$</td></tr>
<tr><td rowspan="3">9</td><td colspan="2">Check for permitted power consumption.</td></tr>
<tr><td>For JC-3xx:</td><td>For JX3-PS1:</td></tr>
<tr><td>$\sum P_{5V} \leq 6W$<br>$\sum P_{24V} \leq 18W$</td><td>$\sum P_{5V} \leq 6W$<br>$\sum P_{24V} \leq 24W$</td></tr>
</table>

| Step | Action |
|---|---|
| ⇨ | The number of JX3-THI2-RTD modules is limited by the power consumption at 5 V:<br><br>For JC-3xx and JX3-PS1:<br>$n = \frac{\sum P_{5V}}{P_{5V}} = \frac{6W}{1.05W} = 5.71 \Rightarrow 5$ |
| **10** | Insert one JX3-PS1 per 5 JX3-THI2-RTD. |

**Engineering the JX3 station - Option 1**

Engineer the JX3 station in our example which is equipped with eleven JX3-AO4 modules and five JX3-THI2-RTD modules as shown below:




| Number | Part | Description |
|---|---|---|
| **1** | JC-3xx | Controller |
| **2** | JX3-AO4 | Modules 1 to 5 supplied by JC-3xx (1) |
| **3** | JX3-PS1 | Power supply module |
| **4** | JX3-AO4 | Modules 6 to 11 supplied by JX3-PS1 (3) |
| **5** | JX3-PS1 | Power supply module |
| **6** | JX3-THI2-RTD | Modules 12 to 16 supplied by JX3-PS1 (5) |

**Engineering the JX3 station - Option 2**

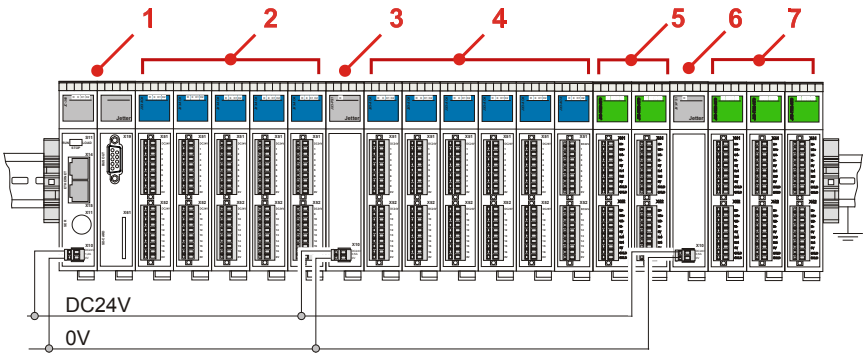
There is also the option of connecting two JX3-THI2-RTD directly to the first JX3-PS1 and to connect the remaining three JX3-THI2-RTD to the second JX3-PS1.

$$\sum P_{5V} = 6 \cdot P_{5V\,JX3\text{-}AO4} + 2 \cdot P_{5V\,JX3\text{-}THI2\text{-}RTD} = 2.1W + 2.1W = 4.2W \leq 6W$$
$$\sum P_{24V} = 6 \cdot P_{24V\,JX3\text{-}AO4} + 2 \cdot P_{24V\,JX3\text{-}THI2\text{-}RTD} = 19.7W + 2.5W = 22.2W \leq 24W$$

Regarding power consumption, you can even connect three JX3-THI2-RTD to the first JX3-PS1.

$$\sum P_{5V} = 6 \cdot P_{5V\,JX3\text{-}AO4} + 3 \cdot P_{5V\,JX3\text{-}THI2\text{-}RTD} = 2.1W + 3.15W = 5.25W \leq 6W$$
$$\sum P_{24V} = 6 \cdot P_{24V\,JX3\text{-}AO4} + 3 \cdot P_{24V\,JX3\text{-}THI2\text{-}RTD} = 19.7W + 3.8W = 23.5W \leq 24W$$

Yet, the maximum number of eight modules which can be connected to a JX3-PS1 will then be exceeded.




| Number | Part | Description |
|---|---|---|
| **1** | JC-3xx | Controller |
| **2** | JX3-AO4 | Modules 1 to 5 supplied by JC-3xx (1) |
| **3** | JX3-PS1 | Power supply module |
| **4** | JX3-AO4 | Modules 6 to 11 supplied by JX3-PS1 (3) |
| **5** | JX3-THI2-RTD | Modules 12 and 13 supplied by a JX3-PS1 (3) |
| **6** | JX3-PS1 | Power supply module |
| **7** | JX3-THI2-RTD | Modules 14 to 16 supplied by JX3-PS1 (6) |

# 4.8 Configuring the JX2 system bus

**Introduction**

This chapter is on configuring the JX2 system bus.

**Number of connectable modules**

The following table shows the maximum number of modules which you can connect in parallel to the JX2 system bus of the JC-360.

| Controller | JX2-I/O modules IP67 modules | JX-SIO CANopen® modules | JX2 slave modules JetMove |
|---|---|---|---|
| JC-360-4 | 23 | 10 | 4 |
| JC-360-8 | 23 | 10 | 8 |
| JC-360 | 23 | 10 | 16 |

**Connectable modules**

The following modules can be connected to the JX2 system bus of a JC-360:

- JX2-I/O modules
- JX2 slave modules
- Servo amplifiers JetMove 1xx, JetMove 2xx, and JetMove 6xx
- IP67-I/O modules LioN-S and LJX7-CSL
- JX-SIO and Smart I/O

**Third-party CANopen® modules**

The following third-party CANopen® modules can be connected to the system bus of the controller JC-360:

- Valve terminals by Festo
- Valve terminals by SMC
- Valve terminals by Bürkert
- I/O system 750 by Wago
- Gateway BWU1821 by Bihl+Wiedemann
- EcoStep drives by Jenaer Antriebstechnik
- EPOS drives by maxon
- Drives by Festo
- Drives by Lenze
- Milan drive by GFC

**Contents**


| Topic | Page |
|---|---|
| Wiring the JX2 system bus | 105 |
| Third-party CANopen® modules | 111 |

# 4.8.1 Wiring the JX2 system bus

**Introduction**

This chapter gives a description of the physical structure of the JX2 system bus.

**Contents**


| Topic | Page |
|---|---|
| Line length and baud rate of the JX2 system bus | 106 |
| JX2 system bus topology | 107 |
| Power supply of JX2-I/O modules | 108 |
| Power supply of JX2 slave modules | 110 |

# Line length and baud rate of the JX2 system bus

**Cable lengths**

The maximum cable length depends on the baud rate used and the number of expansion modules connected to the bus.

| Baud rate | Cable length | Stub length | Total stub length |
|---|---|---|---|
| 1,000 kBaud | 25 m max. | 0.3 m max. | 3 m |
| 500 kBaud | 100 m max. | 1.0 m max. | 39 m |
| 250 kBaud | 200 m max. | 3.0 m max. | 78 m |
| 125 kBaud | 200 m max. | - | - |

**Rules for calculating the stub length**

When engineering the line length, follow the rules listed below:

- Each non-intelligent JX2-I/O module connected to the system bus reduces the maximum line length by 1.0 m
- Each connected intelligent JX2-I/O slave module reduces the maximum line length by 1.0 m
- Each JetMove reduces the maximum line length by 1.0 m
- Each connected IP67-I/O module reduces the maximum line length by 1.0 m

**Baud rate**

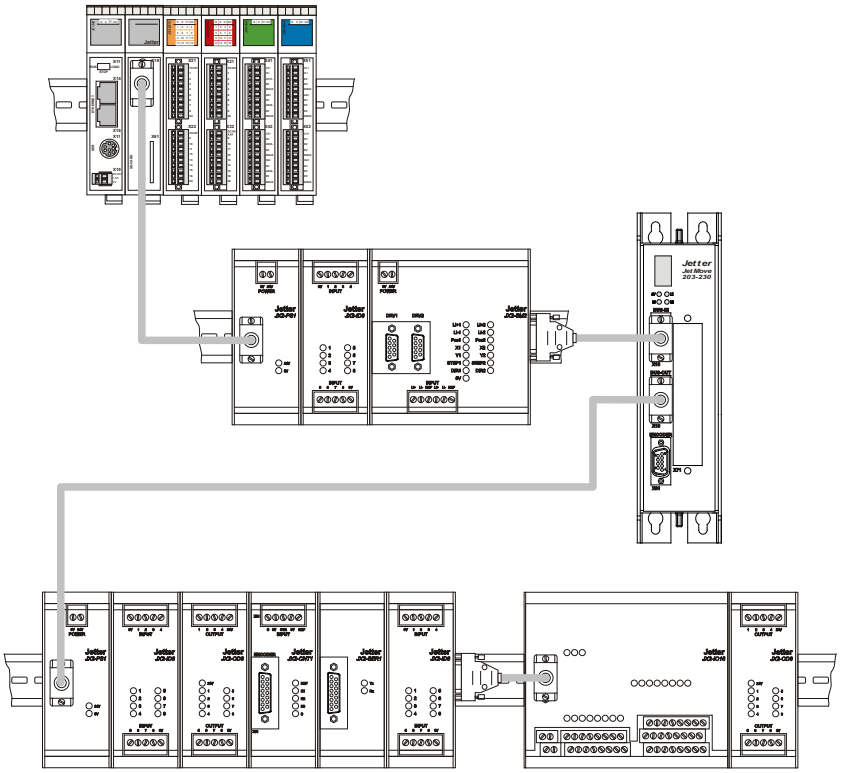
The baud rate setting depends on the number of modules connected to the JX2 system bus:

| JX2-I/O modules<br>JX2 slave modules<br>JetMove | JX-SIO<br>CANopen® modules | 1,000 kBaud | 500 kBaud | 250 kBaud | 125 kBaud |
|---|---|---|---|---|---|
| x | | x | x | x | x |
| | x | x | x | x | x |
| x | x | x | | | x |

## JX2 system bus topology

**Remote arrangement**

The JX2 system bus lets you connect one or several remote modules to a JC-360. The overall distance from the controller may add up to 200 meters.



**Bus terminating resistor**

All JX2 modules are equipped with an internal bus terminating resistor. When the original cable set by Jetter AG is used, bus termination is activated automatically.

**Topology**

The JX2 system bus topology is a line topology. At the end of the line is always a JC-360.

# Power supply of JX2-I/O modules

**JX2-I/O modules**

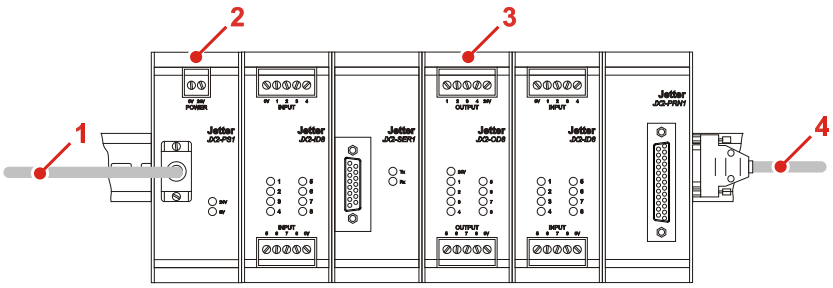
For the following JX2-I/O modules an external power supply is required:

- Digital modules: JX2-ID8, JX2-OD8, JX2-SIM8
- Analog modules: JX2-IA4, JX2-OA2, JX2-OA4
- Other modules: JX2-CNT1, JX2-SER1, JX2-PRN1

The JX2-IO16 does not require an external power supply.

**Power supply of modules using JX2-PS1**

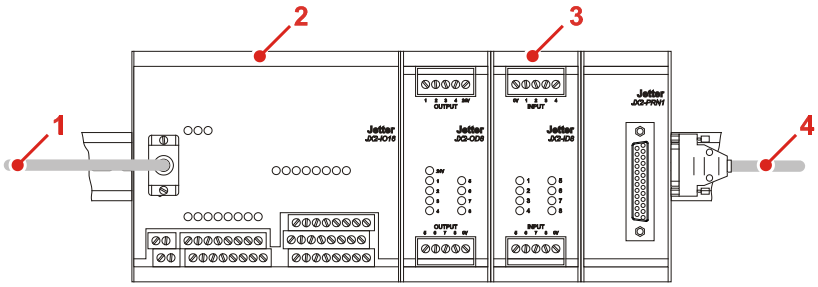
One power supply module JX2-PS1 can directly supply up to five JX2-I/O modules.




| Number | Part | Description |
|---|---|---|
| **1** | IN | JX2 system bus cable connecting to the controller |
| **2** | JX2-PS1 | Power supply module |
| **3** | JX2-I/O | A maximum of five JX2-I/O modules equipped with various interfaces |
| **4** | OUT | JX2 system bus cable leading to additional modules |

**Power supply of modules using JX2-IO16**

One input/output module JX2-IO16 supplies up to three JX2-I/O modules.




| Number | Part | Description |
|---|---|---|
| **1** | IN | JX2 system bus cable connecting to the controller |
| **2** | JX2-IO16 | Input/output module |
| **3** | JX2-I/O | A maximum of three JX2-I/O modules equipped with various interfaces |
| **4** | OUT | JX2 system bus cable leading to additional modules |

# Power supply of JX2 slave modules

**JX2 slave modules**

JX2 slave modules are directly supplied with a voltage of DC 24 V. JX2 slave modules are not capable of supplying other modules with power. The following modules are JX2 slave modules:

- JX2-SM2, JX2-SM1D, JX2-PROFI1, JX2-PID1, JX2-SV1

**Compliance with EMC**

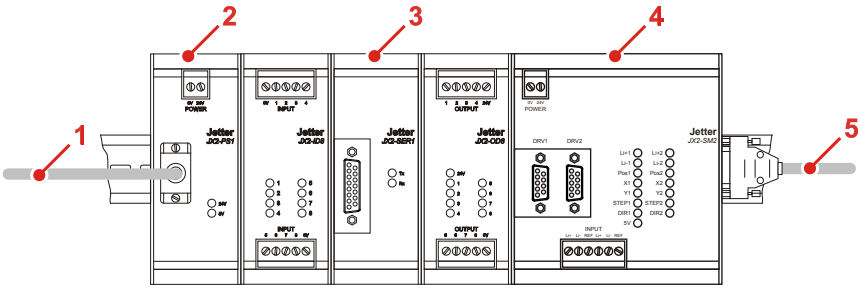
To the left of each JX2 slave module another JX2 module must be installed to ensure EMC and noise immunity. However, this does not apply to JetMoves.

| If... | ... then ... |
|---|---|
| ... the JX2-SM2 module is the first module of a JX2 station, ... | ... install a JX2-PS1 power supply module to the left of it. |

**Heterogeneous JX2 station**

A heterogeneous JX2 station comprises of JX2 slave modules and JX2-I/O modules. Modules of a JX2 station must be installed from left to right in the following order:

- Power supply module JX2-PS1
- JX2-I/O modules
- JX2 slave modules (other than JetMove)




| Number | Part | Description |
|---|---|---|
| **1** | IN | JX2 system bus cable connecting to the controller |
| **2** | JX2-PS1 | Power supply module |
| **3** | JX2-I/O | A maximum of five JX2-I/O modules equipped with various interfaces |
| **4** | JX2 slave | JX2 slave modules with power supply of their own (other than JetMove) |
| **5** | OUT | JX2 system bus cable leading to additional modules |

## 4.8.2 Third-party CANopen® modules

**Introduction**

Third-party CANopen® modules can directly be connected to the JX2 system bus of a JC-360.

**Configuration**

The JX2 system bus needs not be configured. The JC-360 is able to automatically detect and commission connected CANopen®-modules.

**Contents**


| Topic | Page |
|---|---|
| Product description - Module BWU1821 by Bihl+Wiedemann | 112 |
| Product description - ECOSTEP® | 113 |
| Product description - Festo CPV-Direct | 114 |
| Product description - Festo CPX terminals | 115 |
| Product description - Festo CPX-CP interface | 117 |
| Product description - Festo CPX-CMAX--1 | 118 |
| Product description - Festo CPX-CMPX | 119 |
| Product description - Festo MTR-DCI | 120 |
| Product description - Festo SFC-DC | 121 |
| Product description - Festo SFC-LAC | 122 |
| Product description - Festo SFC-LACI | 123 |
| Product description - Lenze 8200 vector | 124 |
| Product description – maxon EPOS | 125 |
| Product description - Milan drive | 126 |
| Product description – SMC EX120 | 127 |
| Product description – SMC EX250 | 128 |
| Product description - WAGO I/O-System 750 | 129 |

# Product description - Module BWU1821 by Bihl+Wiedemann

**BWU1821**

The BWU1821 is a gateway between CANopen® and AS interface.




| Order reference | Description |
|---|---|
| BWU1821 | AS interface CANopen® gateway |

**Restrictions when connected to the JX2 system bus**

| | |
|---|---|
| Number of BWU1821 modules connected to the JX2 system bus of JC-3xx | 1 max. |
| Digital inputs and outputs | 248 max. |
| Analog I/Os | 124 max. |
| I/O module number | 70 or 71 only<br>The BWU1821 module occupies the next 8 module numbers as well. |

**Minimum requirements**

The BWU1821 can be connected to the JX2 system bus of the following controllers and modules by Jetter AG:

| Controller/module | As of version |
|---|---|
| JC-340/JC-350 | V. 1.05.0.00 |
| JC-360/JC-360MC | V. 1.01.0.00 |
| JC-365/JC-365MC | V. 1.21.0.33 |
| JC-24x | V. 3.23 |
| JM-D203-JC24x | V. 1.12.0.00 |
| JX6-SB(-I) | V. 2.18 |

# Product description - ECOSTEP®

**ECOSTEP®**

The servo amplifier ECOSTEP® is a product by Jenaer Antriebstechnik GmbH.



| Order reference | Description |
|---|---|
| 100-xx-000 | Servo amplifier 0.56 kW |
| 200-xx-000 | Servo amplifier 2 kW |

**Restrictions when connected to the JX2 system bus**

| | |
|---|---|
| Number of ECOSTEP® modules connected to the JX2 system bus | 10 max.<br>Additionally limited by the number of axes of the controller |

**Minimum requirements**

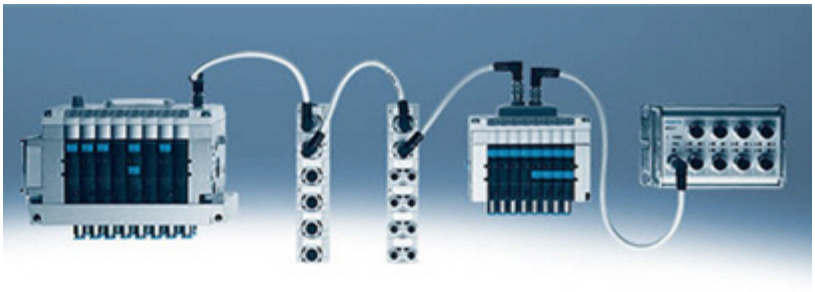
The ECOSTEP® module can be connected to the JX2 system bus of the following controllers and modules by Jetter AG:

| Controller/module | As of version |
|---|---|
| JC-340/JC-350 | V. 1.04.0.00 |
| JC-360/JC-360MC | V. 1.01.0.00 |
| JC-365/JC-365MC | V. 1.21.0.33 |
| JC-24x | V. 3.14 |
| JM-D203-JC24x | V. 1.10.0.00 |
| JX6-SB(-I) | V. 2.18 |

# Product description - Festo CPV-Direct

**CPV-Direct**

The CPV-Direct valve terminal is a product by Festo AG & Co. Additional CP modules can be connected to a CPV-Direct.



| Order reference | Description |
|---|---|
| CPV10-GE-CO2-8 | Valve terminal which directly connects to CANopen® |
| CPV14-GE-CO2-8 | Valve terminal which directly connects to CANopen® |
| CPV18-GE-CO2-8 | Valve terminal which directly connects to CANopen® |

**Restrictions when connected to the JX2 system bus**

| | |
|---|---|
| Number of CPV-Direct modules connected to the JX2 system bus | 10 max. |
| Digital inputs and outputs per CPV-Direct | 64 max. |

**Minimum requirements**

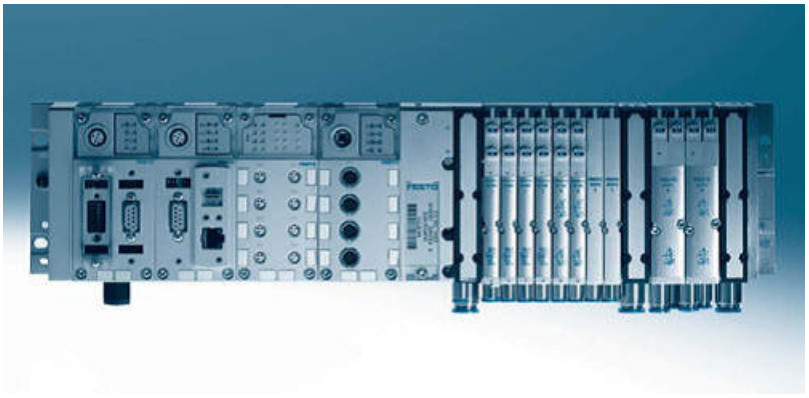
The CPV-Direct valve terminal can directly be connected to the JX2 system bus of the following controllers and modules by Jetter AG:

| Controller/module | As of version |
|---|---|
| JC-340/JC-350 | V. 1.04.0.00 |
| JC-360/JC-360MC | V. 1.01.0.00 |
| JC-365/JC-365MC | V. 1.21.0.33 |
| JC-24x | V. 2.00 |
| JM-D203-JC24x | V. 1.10.0.00 |
| JX6-SB(-I) | V. 2.10 |

# Product description - Festo CPX terminals

**CPX terminal**

A CPX terminal lets you connect digital and analog inputs and outputs, as well as valves of different sizes.



| Order reference | Description |
|---|---|
| CPX-FB14 | Fieldbus node for CANopen® |

**Restrictions when connected to the JX2 system bus**

| | |
|---|---|
| Number of CPX-FB14 modules connected to the JX2 system bus | 10 max. |
| Digital inputs per CPX-FB14 | 64 max. |
| Digital outputs per CPX-FB14 | 64 max. |
| Analog inputs and outputs per CPX-FB14 | 12 max. |
| Digital inputs and outputs per CPX-FB14 and CPX-CP interface | 192 max. |

**Minimum requirements**

The CPX terminal can be connected to the JX2 system bus of the following controllers and modules by Jetter AG:

| Controller/module | As of version |
|---|---|
| JC-340/JC-350 | V. 1.04.0.00 |
| JC-360/JC-360MC | V. 1.01.0.00 |
| JC-365/JC-365MC | V. 1.21.0.33 |
| JC-24x | V. 3.26.0.00 |
| JM-D203-JC24x | V. 1.13.0.00 |
| JX6-SB(-I) | V. 2.21 |

**Restrictions when using the CPX-CP interface**

When using a CPX-CP interface connected to CPX-FB14 the following restrictions have to be observed:

- The CPX-FB14 supports only one CPX-CP interface.
- The CPX-FB14 occupies on the system bus up to three I/O module numbers for CANopen® modules. These I/O module numbers are not available to other CANopen® modules.
- The number of analog I/Os connected to the CPX-FB14 is reduced.

**Restrictions when connecting analog I/Os**

When a CPX-CP interface is used, the number of analog inputs and outputs which can be connected to the CPX-FB14 is reduced accordingly.

| If to the CP string 1 or 2 ... | ... then the number of ... |
|---|---|
| ... CP/CPI output modules are connected, ... | ... analog outputs reduces to 8. |
| ... CP valve terminals are connected, ... | ... analog outputs reduces to 8. |
| ... CP/CPI input modules are connected, ... | ... analog inputs reduces to 8. |

| If to the CP string 3 or 4 ... | ... then the number of ... |
|---|---|
| ... CP/CPI output modules are connected, ... | analog outputs reduces to 4 |
| ... CP valve terminals are connected, ... | analog outputs reduces to 4 |
| ... CP/CPI input modules are connected, ... | ... analog inputs reduces to 4. |

**Restrictions when connecting CANopen® modules**

The maximum amount of CANopen® modules which can be connected to the JX2 system bus is limited when additional I/O module numbers are used. Each additional I/O module number used by CPX-FB14 reduces the number of CANopen® modules by one.

# Product description - Festo CPX-CP interface

**CPX-CP interface**

The CPX-CP interface is for connecting expansion modules of the CP/CPI installation system to the CPX terminal.



| Order reference | Description |
|---|---|
| CPX-CPI | Interface for modules of the CP/CPI installation system |

**Restrictions when connected to the JX2 system bus**

| | |
|---|---|
| Number of CPX-CPI modules on the CPX terminal | 1 |
| Size of the cyclic I/O data | 2 bytes per string<br>8 bytes max. |
| Number of entries in object 0x6100 | 1 entry per string |
| Number of entries in object 0x6300 | 1 entry per string |

**Minimum requirements**

The technology module CPX-CP interface can be connected to the JX2 system bus of the following controllers and modules by Jetter AG:

| Controller/module | As of version |
|---|---|
| JC-340/JC-350 | V. 1.03.0.00 |
| JC-360/JC-360MC | V. 1.03.0.00 |
| JC-365/JC-365MC | V. 1.21.0.33 |
| JC-24x | V. 3.26 |
| JX6-SB(-I) | V. 2.21 |

# Product description - Festo CPX-CMAX--1

**CPX-CMAX-1**

The CPX-CMAX is a servo-pneumatic positioning system for controlling pneumatic drives. It is connected to a CPX terminal and controls the positions of different pneumatic drive systems (linear or rotatory). It is protected to IP65.




| Order reference | Description |
|---|---|
| CPX-CMAX-1 | Servo-pneumatic positioning controller |

**Restrictions when connected to the JX2 system bus**

| | |
|---|---|
| Number of CPX-CMAX-1 modules on the CPX terminal | 4 max. |
| Size of the cyclic I/O data | 8 byte |
| Number of entries in object 0x6100 | 4 |
| Number of entries in object 0x6300 | 4 |

**Minimum requirements**

The technology module CPX-CMAX-1 can be connected to the JX2 system bus of the following controllers by Jetter AG:

| Controller | As of version |
|---|---|
| JC-340/JC-350 | V. 1.10.0.00 |
| JC-360/JC-360MC | V. 1.10.0.00 |
| JC-365/JC-365MC | V. 1.21.0.33 |

**Application Note**

For more information on how to operate these modules along with a JC-360, refer to the following application notes:

| Modules | Document |
|---|---|
| MTR/SFC | Festo_apn042_xxx_Festo_Electrical_Motor_Controllers.pdf |
| CPX | Festo_apn043_xxx_CPX_Technologiemodule.pdf |

# Product description - Festo CPX-CMPX

**CPX-CMPX**

The CPX-CMPX is an electronic end-position controller for pneumatic drives. It allows fast travel between the mechanical end stops, stopping gently and without impact in the end position.




| Order reference | Description |
|---|---|
| CPX-CMPX | Electronic end-position controller |

**Restrictions when connected to the JX2 system bus**

| | |
|---|---|
| Number of CPX-CMPX modules on the CPX terminal | 5 max. |
| Size of cyclic I/O data | 6 byte |
| Number of entries in object 0x6100 | 3 |
| Number of entries in object 0x6300 | 3 |

**Minimum requirements**

The technology module CPX-CMPX can be connected to the JX2 system bus of the following controllers by Jetter AG:

| Controller | As of version |
|---|---|
| JC-340/JC-350 | V. 1.10.0.00 |
| JC-360/JC-360MC | V. 1.10.0.00 |
| JC-365/JC-365MC | V. 1.21.0.33 |

**Application Note**

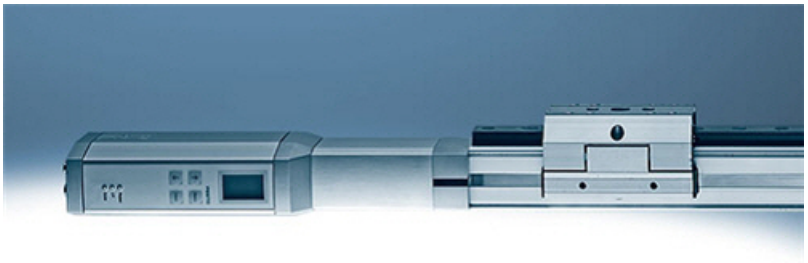
For more information on how to operate these modules along with a JC-360, refer to the following application notes:

| Modules | Document |
|---|---|
| MTR/SFC | Festo_apn042_xxx_Festo_Electrical_Motor_Controllers.pdf |
| CPX | Festo_apn043_xxx_CPX_Technologiemodule.pdf |

# Product description - Festo MTR-DCI

**MTR-DCI**

The motor controller MTR-DCI is a product by Festo AG & Co. The motor controller MTR-DCI is an innovative motor with integrated power electronics and has been designed for positioning tasks.



| Order reference | Description |
|---|---|
| MTR-DCI-32-xxx-CO | Frame size 32 with CANopen® interface |
| MTR-DCI-42-xxx-CO | Frame size 42 with CANopen® interface |
| MTR-DCI-52-xxx-CO | Frame size 52 with CANopen® interface |
| MTR-DCI-62-xxx-CO | Frame size 62 with CANopen® interface |

**Restrictions when connected to the JX2 system bus**

| | |
|---|---|
| Number of MTR-DCI units connected to the JX2 system bus | 10 max.<br>Additionally limited by the number of axes of the controller |

**Minimum requirements**

The motor controller MTR-DCI can be connected to the JX2 system bus of the following controllers by Jetter AG:

| Controller | As of version |
|---|---|
| JC-340/JC-350 | V. 1.10.0.00 |
| JC-360/JC-360MC | V. 1.10.0.00 |
| JC-365/JC-365MC | V. 1.21.0.33 |

**Application Note**

For more information on how to operate these modules along with a JC-360, refer to the following application notes:

| Modules | Document |
|---|---|
| MTR/SFC | Festo_apn042_xxx_Festo_Electrical_Motor_Controllers.pdf |
| CPX | Festo_apn043_xxx_CPX_Technologiemodule.pdf |

# Product description - Festo SFC-DC

**SFC-DC**

The motor controller SFC-DC is a product by Festo AG & Co. The motor controller SFC-DC is a positioning and position feedback controller for electric mini slides SLTE.



| Order reference | Description |
|---|---|
| SFC-DC-xxx-H0-CO | Without control panel with CANopen® interface |
| SFC-DC-xxx-H2-CO | With control panel with CANopen® interface |

**Restrictions when connected to the JX2 system bus**

| | |
|---|---|
| Number of SFC-DC modules connected to the JX2 system bus | 10 max.<br>Additionally limited by the number of axes of the controller |

**Minimum requirements**

The motor controller SFC-DC can be connected to the JX2 system bus of the following controllers by Jetter AG:

| Controller | As of version |
|---|---|
| JC-340/JC-350 | V. 1.10.0.00 |
| JC-360/JC-360MC | V. 1.10.0.00 |
| JC-365/JC-365MC | V. 1.21.0.33 |

**Application Note**

For more information on how to operate these modules along with a JC-360, refer to the following application notes:

| Modules | Document |
|---|---|
| MTR/SFC | Festo_apn042_xxx_Festo_Electrical_Motor_Controllers.pdf |
| CPX | Festo_apn043_xxx_CPX_Technologiemodule.pdf |

# Product description - Festo SFC-LAC

**SFC-LAC**

The motor controller SFC-LAC is a product by Festo AG & Co. The motor controller SFC-LAC is used as positioning and position feedback controller for linear handling axes HME.



| Order reference | Description |
|---|---|
| SFC-LAC-xxx-H0-CO | Without control panel with CANopen® interface |
| SFC-LAC-xxx-H2-CO | With control panel with CANopen® interface |

**Restrictions when connected to the JX2 system bus**

| | |
|---|---|
| Number of SFC-LAC modules connected to the JX2 system bus | 10 max.<br>Additionally limited by the number of axes of the controller |

**Minimum requirements**

The motor controller SFC-LAC can be connected to the JX2 system bus of the following controllers by Jetter AG:

| Controller | As of version |
|---|---|
| JC-340/JC-350 | V. 1.10.0.00 |
| JC-360/JC-360MC | V. 1.10.0.00 |
| JC-365/JC-365MC | V. 1.21.0.33 |

**Application Note**

For more information on how to operate these modules along with a JC-360, refer to the following application notes:

| Modules | Document |
|---|---|
| MTR/SFC | Festo_apn042_xxx_Festo_Electrical_Motor_Controllers.pdf |
| CPX | Festo_apn043_xxx_CPX_Technologiemodule.pdf |

# Product description - Festo SFC-LACI

**SFC-LACI**

The motor controller SFC-LACI is a product by Festo AG & Co. The motor controller SFC-LACI is used as positioning and position feedback controller for the following electric drives: DNCE-...-LAS and DFME-...-LAS.



| Order reference | Description |
|---|---|
| SFC-LACI-xxx-H0-CO | Without control panel with CANopen® interface |
| SFC-LACI-xxx-H2-CO | With control panel with CANopen® interface |

**Restrictions when connected to the JX2 system bus**

| | |
|---|---|
| Number of SFC-LACI modules connected to the JX2 system bus | 10 max.<br>Additionally limited by the number of axes of the controller |

**Minimum requirements**

The motor controller SFC-LACI can be connected to the JX2 system bus of the following controllers by Jetter AG:

| Controller | As of version |
|---|---|
| JC-340/JC-350 | V. 1.10.0.00 |
| JC-360/JC-360MC | V. 1.10.0.00 |
| JC-365/JC-365MC | V. 1.21.0.33 |

# Product description - Lenze 8200 vector

**Lenze 8200 vector**

The frequency inverter 8200 vector is a product by Lenze SE. The frequency inverter can directly be connected to the JX2 system bus via communications module or function module.



| Order reference | Description |
|---|---|
| 2175 CANopen®/DeviceNet | Communications module for CANopen® |
| CANopen® PT | Function module for CANopen® |

**Restrictions when connected to the JX2 system bus**

| | |
|---|---|
| Number of 8200 vector modules connected to JX2 system bus | 10 max. |

**Minimum requirements**

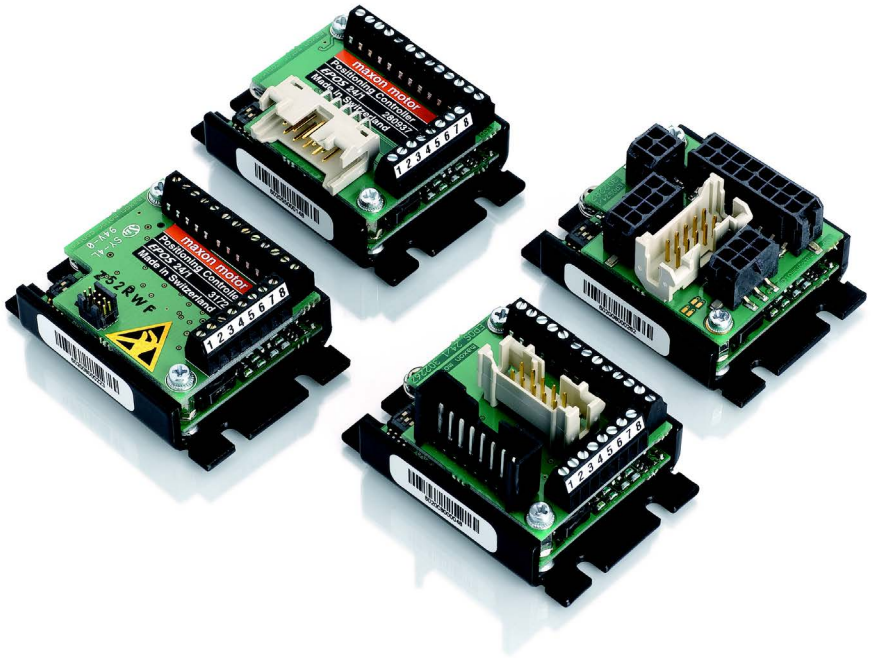
The frequency converter 8200 vector can directly be connected to the JX2 system bus of the following controllers and modules by Jetter AG:

| Controller/module | As of version |
|---|---|
| JC-340/JC-350 | V. 1.05.0.00 |
| JC-360/JC-360MC | V. 1.01.0.00 |
| JC-365/JC-365MC | V. 1.21.0.33 |
| JC-24x | V. 3.10 |
| JM-D203-JC24x | V. 1.10.0.00 |
| JX6-SB(-I) | V. 2.10 |

# Product description – maxon EPOS

**maxon EPOS**

The positioning controller maxon EPOS is a product by maxon motor AG. It can directly be connected to the JX2 system bus.




| Order reference | Description |
|---|---|
| EPOS 24/1 | Positioning controller 24 V/1 A |
| EPOS 24/5 | Positioning controller 24 V/5 A |

**Restrictions when connected to the JX2 system bus**

| | |
|---|---|
| Number of EPOS connected to the JX2 system bus | 10 max. |

**Minimum requirements**

The positioning controller EPOS can be connected to the JX2 system bus of the following controllers and modules by Jetter AG:

| Controller/module | As of version |
|---|---|
| JC-340/JC-350 | V. 1.04.0.00 |
| JC-360/JC-360MC | V. 1.01.0.00 |
| JC-365/JC-365MC | V. 1.21.0.33 |
| JC-24x | V. 3.15 |
| JM-D203-JC24x | V. 1.10.0.00 |
| JX6-SB(-I) | V. 2.14 |

# Product description - Milan drive

**Milan drive**

The servo drive Milan drive is a product by GFC Antriebssysteme GmbH.



| Order reference | Description |
|---|---|
| MI 1.5 / 075 | Milan drive |
| MI 2 / 090 | Milan drive |
| MI 4 / 110 | Milan drive |
| MDA 35.1 | Milan drive Advanced 0.47 W |
| MDA 56.1 | Milan drive Advanced 0.78 W |
| MDA 63.1 | Milan drive Advanced 1.26 W |

**Restrictions when connected to the JX2 system bus**

| | |
|---|---|
| Number of Milan drives connected to JX2 system bus | 10 max.<br>Additionally limited by the number of axes of the controller |

**Minimum requirements**

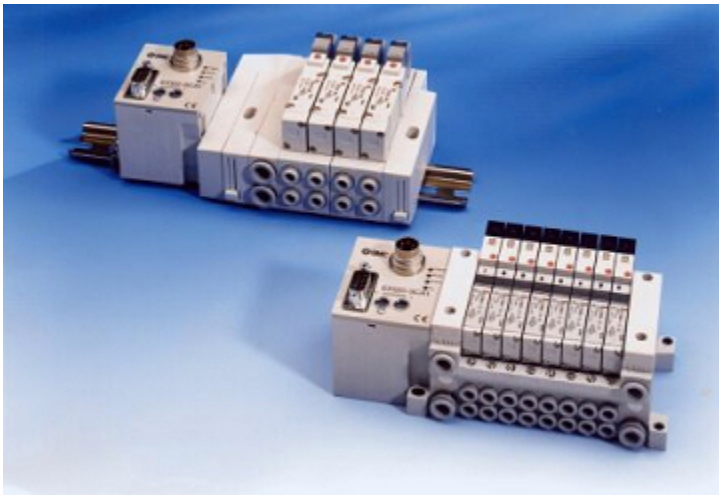
The Milan drive can be connected to the JX2 system bus of the following controllers and modules by Jetter AG:

| Controller/module | As of version |
|---|---|
| JC-340/JC-350 | V. 1.04.0.00 |
| JC-360/JC-360MC | V. 1.01.0.00 |
| JC-365/JC-365MC | V. 1.21.0.33 |
| JC-24x | V. 3.23 |
| JM-D203-JC24x | V. 1.13.0.00 |
| JX6-SB(-I) | V. 2.18 |

# Product description – SMC EX120

**SI unit EX120**

The SI unit EX120 is a product by SMC Pneumatik GmbH. A SI unit EX120 allows connection of valves of different sizes.



| Order reference | Description |
|---|---|
| EX120-SCA1 | SI unit with the CANopen® interface |
| EX121-SCA1 | SI unit with the CANopen® interface |
| EX122-SCA1 | SI unit with the CANopen® interface |

**Restrictions when connected to the JX2 system bus**

| | |
|---|---|
| Number of SI units EX120 connected to the JX2 system bus | 10 max. |
| Digital outputs/valves per EX120 | 64 max. |

**Minimum requirements**

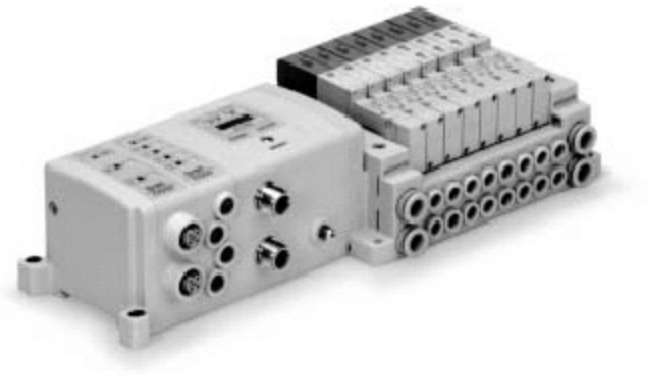
The SI unit EX120 can be connected to the JX2 system bus of the following controllers and modules by Jetter AG:

| Controller/module | As of version |
|---|---|
| JC-340/JC-350 | V. 1.05.0.00 |
| JC-360/JC-360MC | V. 1.01.0.00 |
| JC-365/JC-365MC | V. 1.21.0.33 |
| JC-24x | V. 2.14 |
| JM-D203-JC24x | V. 1.10.0.00 |
| JX6-SB(-I) | V. 2.12 |

# Product description – SMC EX250

**SI unit EX250**

The SI unit EX250 is a product by SMC Pneumatik GmbH. A SI unit EX250 allows connection of digital inputs and valves of different sizes.



| Order reference | Description |
|---|---|
| EX250-SCA1 | SI unit with the CANopen® interface |

**Restrictions when connected to the JX2 system bus**

| | |
|---|---|
| Number of EX250 SI units connected to the JX2 system bus | 10 max. |
| Digital inputs and outputs/valves per EX250 | 64 max. |

**Minimum requirements**

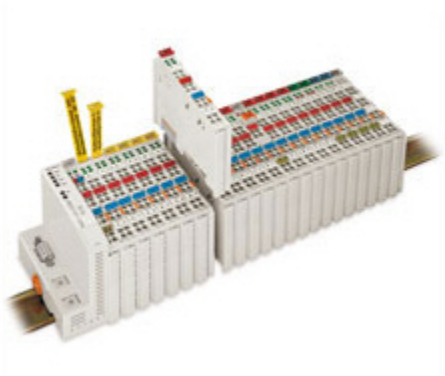
The SI unit EX250 can be connected to the JX2 system bus of the following controllers and modules by Jetter AG:

| Controller/module | As of version |
|---|---|
| JC-340/JC-350 | V. 1.04.0.00 |
| JC-360/JC-360MC | V. 1.01.0.00 |
| JC-365/JC-365MC | V. 1.21.0.33 |
| JC-24x | V. 2.14 |
| JM-D203-JC24x | V. 1.10.0.00 |
| JX6-SB(-I) | V. 2.12 |

# Product description - WAGO I/O-System 750

**I/O system 750**

The I/O system 750 is a product by WAGO Kontakttechnik GmbH. An I/O system 750 allows a great variety of modules to be connected to it.



| Order reference | Description |
|---|---|
| 750-337 | CANopen® field bus coupler, connection to MSS |
| 750-338 | CANopen® field bus coupler, Sub-D connection |

**Restrictions when connected to the JX2 system bus**

| | |
|---|---|
| Amount of I/O-Systems 750 on the JX2 system bus | 10 max. |
| Digital inputs and outputs per I/O-System 750 | 64 max. |
| Analog inputs and outputs per I/O-System 750 | 12 max. |

**Minimum requirements**

The I/O-System can be connected to the JX2 system bus of the following controllers and modules by Jetter AG:

| Controller/module | As of version |
|---|---|
| JC-340/JC-350 | V. 1.04.0.00 |
| JC-360/JC-360MC | V. 1.01.0.00 |
| JC-365/JC-365MC | V. 1.21.0.33 |
| JC-24x | V. 3.20 |
| JM-D203-JC24x | V. 1.10.0.00 |
| JX6-SB(-I) | V. 2.16 |

# 4.9 Connecting displays and HMIs

**Introduction**

This chapter covers the connection of alphanumeric displays and HMIs to the JC-360.

**Contents**


| Topic | Page |
|---|---|
| Overview of displays and HMIs | 131 |
| Connecting a display or HMI | 132 |
| Connecting several displays or HMIs: Multi-display mode | 133 |
| Multi-display mode - Wiring | 134 |
| Interface cable JC-DK-Xm | 136 |
| Interface cable KAY_0386-xxxx | 138 |
| Interface cable KAY_0533-0025 | 140 |

# Overview of displays and HMIs

**List of displays and HMIs**

The following table lists the alphanumeric displays HMIs by Jetter AG which you can connect to the JC-360.

| Order reference | Display | Keys | Interface cable |
|---|---|---|---|
| **LCD 16** | 4 lines of 20 characters each | ■ 5 function keys with LED<br>■ Can be expanded by a keyboard module NUM25 | JC-DK-Xm |
| **LCD 23** | 2 lines of 24 characters each | ■ Cursor left<br>■ Cursor right<br>■ ENTER ([↵]) | JC-DK-Xm |
| **LCD 27** | 2 lines of 24 characters each | ■ 5 function keys<br>■ Cursor keypad<br>■ Clear<br>■ ENTER ([↵]) | JC-DK-Xm |
| **LCD 34** | 2 lines of 24 characters each | ■ 5 function keys<br>■ Numeric keypad | JC-DK-Xm |
| **LCD 52** | 4 lines of 16 characters each | ■ 6 function keys<br>■ Numeric keypad | KAY-0533-0025 |
| **LCD 54** | 4 lines of 16 characters each | ■ 8 function keys<br>■ Numeric keypad<br>■ Emergency stop | KAY-0533-0025 |
| **LCD 54Z** | 4 lines of 16 characters each | ■ 8 function keys<br>■ Numeric keypad<br>■ Emergency stop<br>■ Two-hand control device | KAY-0533-0025 |
| **LCD 60** | 2 lines of 40 characters each | ■ 8 function keys with LED<br>■ Numeric keypad | KAY-0386-xxxx |
| **LCD 110** | 4 lines of 20 characters each | ■ 8 function keys with LED<br>■ Numeric keypad | JC-DK-Xm |

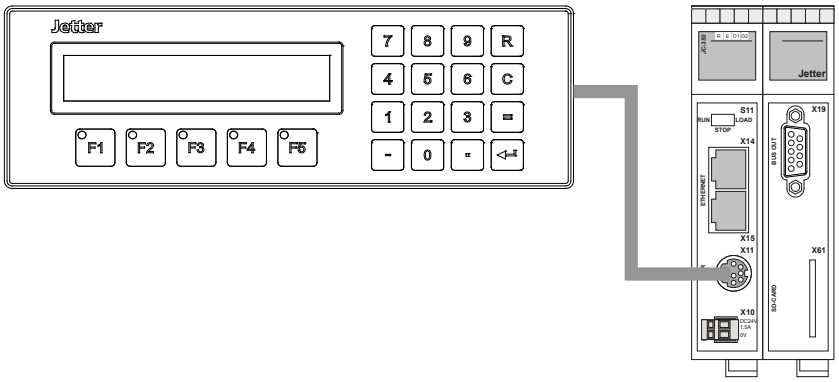
## Connecting a display or HMI

**Connecting a display or HMI**

There are prefabricated cables for connecting a display or HMI made by Jetter AG to a JC-360. We recommend to use the prefabricated cables listed in the above table.

**Interface**

Connect the display or HMI via serial interface to jack X11. To do so, apply connections to interface standard RS-422.




**Restrictions**

Irrespective of the fact that various hardware drivers have been implemented, only one hardware interface is available.

This means:

While, for example, communication via RS-422 is taking place, simultaneous and independent communication via RS-232 is not possible.

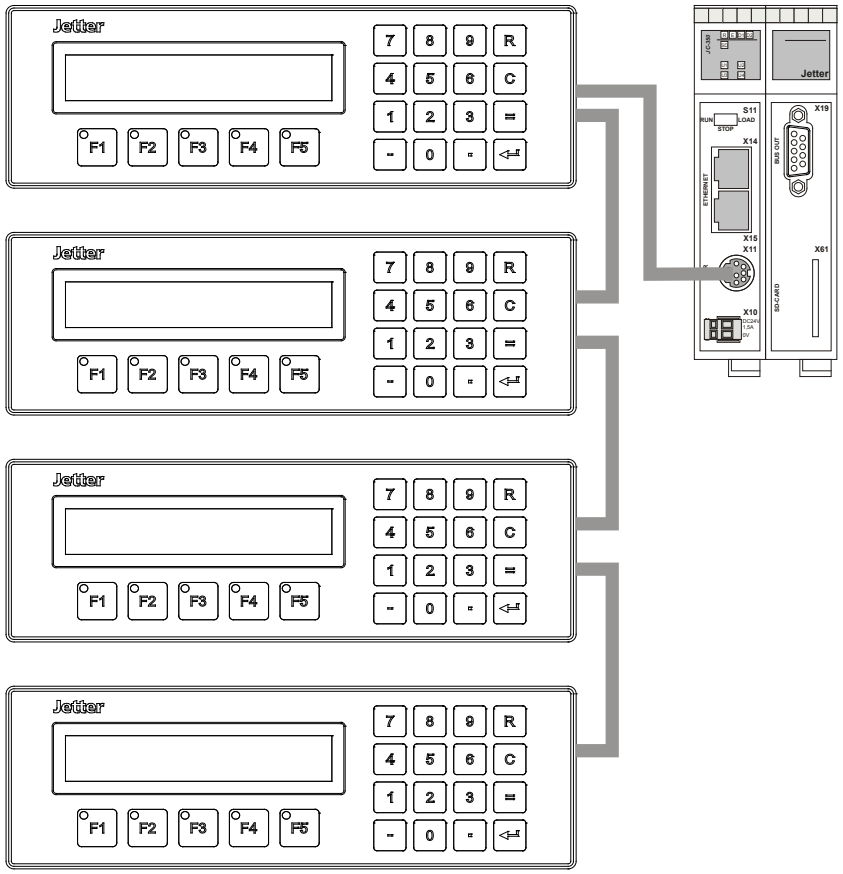
## Connecting several displays or HMIs: Multi-display mode

**Introduction**

Multi-display mode allows a JC-360 to be operated with up to four alphanumeric displays and HMIs connected to one of the serial interfaces. When doing so, the various HMIs display the same or different texts and/or variable contents.

**Interface**

Connect the display or HMI via serial interface to X11. This port supports the interface standard RS-422.




**Restrictions**

Irrespective of the fact that various hardware drivers have been implemented, only one hardware interface is available.

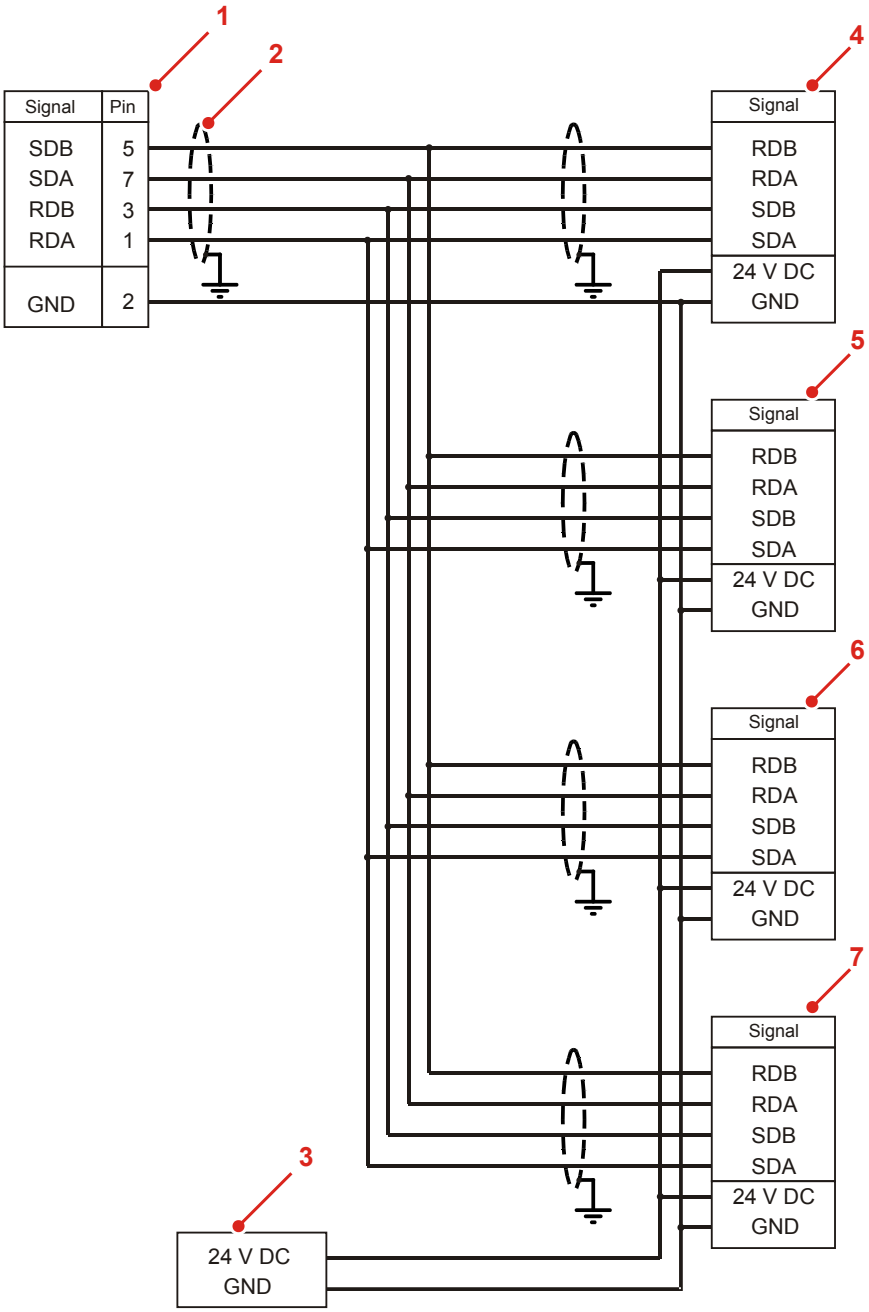
This means:

While, for example, communication via RS-422 is taking place, simultaneous and independent communication via RS-232 is not possible.

## Multi-display mode - Wiring

**Wiring**

There are no prefabricated cables for connecting several displays or HMIs to a controller. Assemble the cables for multi-display mode according to the following block diagram:



| Number | Part | Description |
|---|---|---|
| **1** | Jack X11 | Serial interface of the JC-360 |
| **2** | Shielding | Use shielded cables both ends of which are connected to the metallized housing of the connector. |
| **3** | Power supply | If several displays and HMIs are used, each of them must be individually connected to the power supply. |
| **4 ... 7** | Terminals | Interfaces of displays and HMIs |

**Cable specification**

The following minimum requirements apply to cable sets:

| Parameter | Description |
|---|---|
| Core cross-sectional area | 0.14 $mm^2$ |
| Maximum cable length | 100 m |
| Shielding | Complete shielding, no paired shielding |

# Interface cable JC-DK-Xm

**Introduction**

The interface cable JC-DK-Xm lets you connect displays or HMIs to the JC-360.

**Male connector specification (controller end)**

For information on connector specification refer to the list below:

| | |
|---|---|
| Type | 8-pin male MiniDIN connector |
| Manufacturer | KYCON |
| Item | KMDLA - 8P |
| Recommended core cross section | 0.128 … 0.051 $mm^2$ |

**Male connector specification (HMI end)**

For information on connector specification refer to the list below:

| | |
|---|---|
| Type | 15-pin male SUB-D connector in metallized housing (quality grade 3) |
| Manufacturer | Various manufacturers |
| Recommended core cross section | 0.25 … 0.128 $mm^2$ |

**Specifications of connecting cable**

For information on cable specification refer to the list below:

| | |
|---|---|
| Number of cores | 6 |
| Recommended core cross section | 0.14 $mm^2$ |
| Maximum cable length | 400 m |

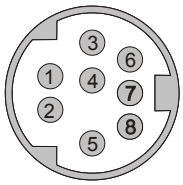
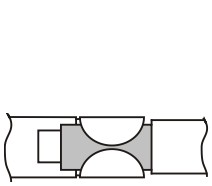
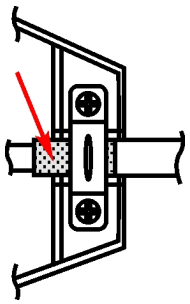
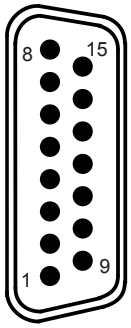
**Cable shielding**

- Complete shielding, no paired shielding
- The shield must be connected to the connector housings on both ends of the cable with the greatest possible surface area.

  Place the shield as follows:
  - Connect the shield by its entire perimeter.
  - Clamp it under a strain relief.
  - Wrap it with copper foil.

**Wiring diagram**

The interface cable JC-DK-Xm is wired as follows:

| Controller | Shielding | | HMI |
|---|---|---|---|
| SER/X11   |   Connect shield with the greatest possible surface area! A metallized housing is required | |   |
| **Pin** | **Signal** | | **Pin** |
| 6 | +24 V DC | | 15 |
| 2 | GND | | 12 |
| 5 | SDB | RDB | 6 |
| 7 | SDA | RDA | 7 |
| 3 | RDB | SDB | 4 |
| 1 | RDA | SDA | 5 |

**Available lengths:**

The interface cable JC-DK-Xm is prefabricated and available in the following lengths:

| Item no. | Item | Description |
|---|---|---|
| 60860011 | Cable assy # 192 2.5M | JetControl to HMI with 15-pin Sub-D, length 2.5 m |
| 60860012 | Cable assy # 193 5M | JetControl to HMI with 15-pin Sub-D, length 5 m |
| 60870894 | Cable assy # 192 7M | JetControl to HMI with 15-pin Sub-D, length 7 m |
| 60872142 | Cable assy # 192 10M | JetControl to HMI with 15-pin Sub-D, length 10 m |
| 60872884 | Cable assy # 192 15M | JetControl to HMI with 15-pin Sub-D, length 15 m |

# Interface cable KAY_0386-xxxx

**Introduction**

The interface cable KAY_0386-xxxx lets you connect HMIs of the type LCD 60 to the JC-360.

**Male connector specification (controller end)**

For information on connector specification refer to the list below:

| | |
|---|---|
| Type | 8-pin male MiniDIN connector |
| Manufacturer | KYCON |
| Item | KMDLA - 8P |
| Recommended core cross section | 0.128 … 0.051 $mm^2$ |

**Female connector specification (HMI end)**

For information on female connector specification refer to the list below:

| | |
|---|---|
| Type | 15-pin female Sub-D connector in metallized housing (quality grade 3). |
| Manufacturer | Various manufacturers |
| Recommended core cross section | 0.25 … 0.128 $mm^2$ |

**Specifications of connecting cable**

For information on cable specification refer to the list below:

| | |
|---|---|
| Number of cores | 5 |
| Recommended core cross section | 0.14 $mm^2$ |
| Maximum cable length | 400 m |

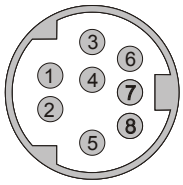
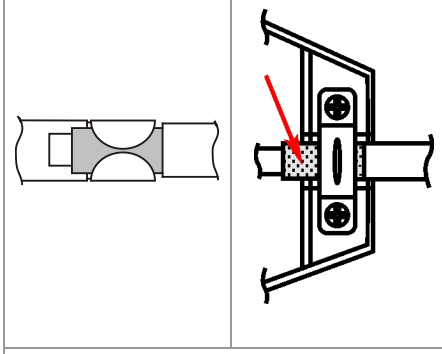
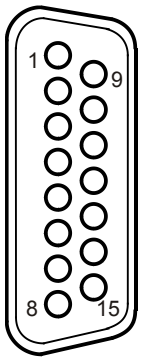
**Cable shielding**

- Complete shielding, no paired shielding
- The shield must be connected to the connector housings on both ends of the cable with the greatest possible surface area.

  Place the shield as follows:
  - Connect the shield by its greatest possible surface area.
  - Clamp it under a strain relief.
  - Wrap it with copper foil.

**Wiring diagram**

The interface cable KAY_0386-xxxx is wired as follows:

| Controller | Shielding | | LCD 60 |
|---|---|---|---|
| SER/X11   |  Connect shield with the greatest possible surface area! A metallized housing is required | | COM 2   |
| **Pin** | **Signal** | | **Pin** |
| 2 | GND | | 5 |
| 5 | SDB | RDB | 13 |
| 7 | SDA | RDA | 12 |
| 3 | RDB | SDB | 15 |
| 1 | RDA | SDA | 14 |
| | | Short-circuited | 11 |
| | | | 10 |

**Available lengths:**

The interface cable KAY_0386-xxxx is prefabricated and available in the following lengths:

| Item no. | Item | Description |
|---|---|---|
| 60864359 | KAY_0386-0250 | JetControl to LCD 60 with 15-pin Sub-D, length 2.5 m |
| 60864360 | KAY_0386-0500 | JetControl to LCD 60 with 15-pin Sub-D, length 5 m |

# Interface cable KAY_0533-0025

**Introduction**

The interface cable KAY_0533-0025 lets you connect HMIs of the type LCD 52, LCD 54 and LCD 54Z to the JC-360.

**Male connector specification (controller end)**

For information on connector specification refer to the list below:

| | |
|---|---|
| Type | 8-pin male MiniDIN connector |
| Manufacturer | KYCON |
| Item | KMDLA - 8P |
| Recommended core cross section | 0.128 … 0.051 $mm^2$ |

**Female connector specification (HMI end)**

For information on female connector specification refer to the list below:

| | |
|---|---|
| Type | 15-pin female Sub-D connector in metallized housing (quality grade 3). |
| Manufacturer | Various manufacturers |
| Recommended core cross section | 0.25 … 0.128 $mm^2$ |

**Specifications of connecting cable**

For information on cable specification refer to the list below:

| | |
|---|---|
| Number of cores | 6 |
| Recommended core cross section | 0.14 $mm^2$ |
| Cable length | 0.25 m |

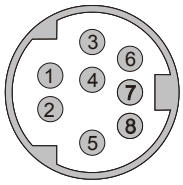
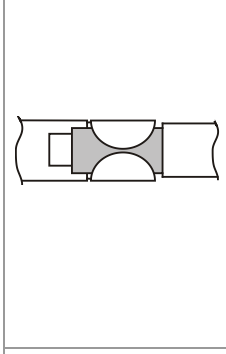
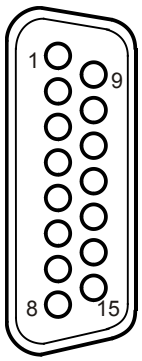
**Cable shielding**

- Complete shielding, no paired shielding
- The shield must be connected to the connector housings on both ends of the cable with the greatest possible surface area.

  Place the shield as follows:
  - Connect the shield by its greatest possible surface area.
  - Clamp it under a strain relief.
  - Wrap it with copper foil.

**Wiring diagram**

The interface cable KAY_0533-0025 is wired as follows:

| Controller | Shielding | | LCD 52/54/54Z |
|---|---|---|---|
| SER/X11   |  Connect shield with the greatest possible surface area! A metallized housing is required | |   |
| **Pin** | **Signal** | | **Pin** |
| 6 | +24 V DC | | 4 |
| 2 | GND | | 7 |
| 5 | SDB | RDB | 10 |
| 7 | SDA | RDA | 11 |
| 3 | RDB | SDB | 12 |
| 1 | RDA | SDA | 13 |

**Available lengths:**

The interface cable KAY_0533-0025 is prefabricated and available in the following lengths:

| Item no. | Item | Order reference |
|---|---|---|
| 60864897 | KAY_0533-0025 | JetControl to LCD 52/54 with 15-pin Sub-D, length 0.25 m |

# 5 Initial commissioning

**Purpose of this chapter**

The first part of this chapter gives a compact description of the initial commissioning of the JC-360 and covers the following functions:

- Creation and execution of a program which increments a variable.

This chapter covers the following topics on initial commissioning of the peripheral module JX3-DIO16 connected with the JC-360:

- Configuring the hardware of a JX3 station and installing it
- Configuring the software in JetSym
- Creating and executing a program to set and reset output 9 of the peripheral module JX3-DIO16

**Prerequisites**

For initial commissioning the JC-360, the following prerequisites must be fulfilled:

- The controller is connected to a PC via Ethernet.
- The programming tool JetSym 4.2 or higher is installed on the PC.
- Mode selector S11 is in *STOP* position.

**Contents**


| Topic | Page |
|---|---|
| Preparatory work for initial commissioning | 144 |
| Initial commissioning of a JC-360 | 145 |
| Configuring error states | 147 |
| Configuration in JetSym | 148 |

# Preparatory work for initial commissioning

**Ethernet connection with the controller**

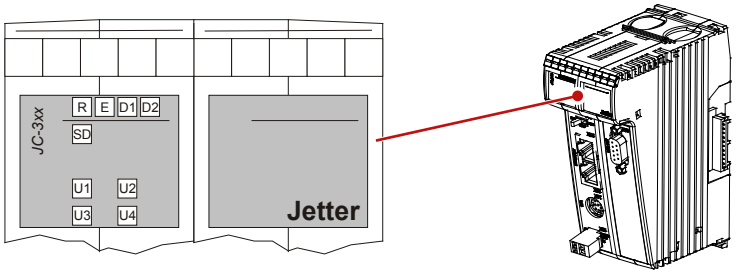
The JC-360 in delivered condition has got IP address 192.168.1.1. Configure the Ethernet interface of your PC so that it communicates with the controller via this IP address.

**Behavior after power-up**

If the mode selector is in position *STOP* when the controller is powered-up, the application program will not be launched.

**Status of the LEDs**

Following a correct commissioning, the LEDs are lit as follows:

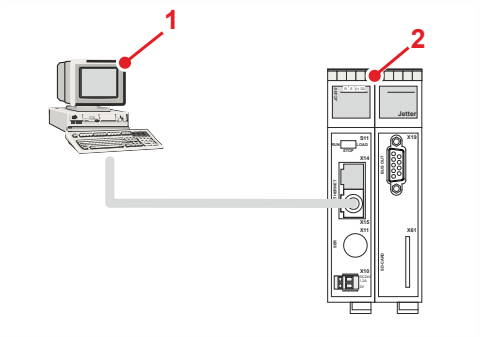



| LED | State | Description |
|---|---|---|
| **R** | 1Hz | Logic voltage supply is OK; application program has been stopped. |
| **E** | OFF | No error |
| **D1** | OFF | No error |
| **D2** | OFF | No error |
| **SD** | OFF | The controller does not access the SD card. |
| **U1 - U4** | OFF | LEDs which are programmed depending on the application |

# Initial commissioning of a JC-360

## Configuration

The initial commissioning is based on the following configuration:




| Number | Part | Description |
|---|---|---|
| 1 | PC | Programming system |
| 2 | JC-360 | Controller |

## Creating an application program

To create and check the program, proceed as follows:

| Step | Action |
|---|---|
| 1 | Start the programming tool JetSym. |
| 2 | Create a new project. |
| 3 | In JetSym, start the Hardware Manager either by clicking on the tab **Hardware** with the mouse (keyboard shortcut **Alt + 5**).<br>Open the window for controller configuration by double-clicking the folder **CPU** in the Hardware Manager. Select the controller type JC-360. |
| 4 | Enter the following information:<br>■ Installed OS version<br>■ **Ethernet** interface type<br>■ IP address |
| 5 | Open the program editor. |
| 6 | Enter the program specifications. |
| 7 | Compile the program by clicking menu item **Build** in menu **Build** (keyboard shortcut **F7**). |
| 8 | Load up the project to the controller by clicking menu item **Download** in menu **Build** (keyboard shortcut **Strg+5**). |
| 9 | Open a setup window. |
| 10 | Enter the variable name (Count). |
| 11 | Activate the setup. |

## JetSym online help

For a detailed description of the JetSym programming software, refer to the JetSym online help.

**JetSym STX program**

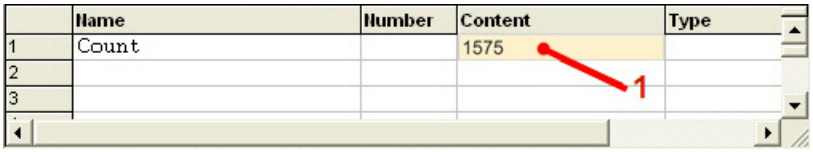
The following program increments the content of a variable by one every 2 seconds:

```
Var
    Count:    Int;
End_Var;

Task Increment Autorun
    Loop
        Inc(Count);
        Delay(T#2s);
    End_Loop;
End_Task;
```

**Setup pane**

The JetSym setup window displays the content of the variable:




| Number | Description | Function |
|---|---|---|
| **1** | Present content of the variable | The content of the variable is incremented by one every 2 seconds. |

# Configuring error states

**Introduction**

JetSym lets you configure the JC-360.

**Prerequisites**

The following requirements must be satisfied:

- JetSym has been installed on the PC used.
- JetSym has been licensed (see online help in JetSym).
- Limitations to be taken into account when engineering a JX3 station have been observed.

**Commissioning steps**

To successfully commission the controller and the JX3 modules connected to it, proceed as follows:

| Step | Action |
|---|---|
| **1** | Make sure that the power supply is disconnected! |
| **2** | Add the required JX3 modules to form the JX3 station. When engineering a JX3 station, consider the limitations applying to its setup.<br>Applying the **JX3 system bus configurator** (see page 92) |
| **3** | Use an Ethernet cable to connect the controller JC-3xx and the PC. |
| **4** | Set an IP address at the controller JC-3xx.<br>For more information, refer to chapter **IP configuration** (see page 71). |
| **5** | On the PC, set an IP address which differs from the IP address of the controller.<br>Example: The controller has got IP address 192.168.1.1. Thus, for the PC, IP address 192.168.1.20 can be used.<br>Make sure that the first three elements of the IP addresses are identical. |
| **6** | Switch on the power supply for the JX3 station. |
| **7** | Launch JetSym. Then configure the JX3 station following the sample program.<br>For more information on this sample program, refer to chapter **Configuration in JetSym** (see page 148). |
| **8** | Configure the JX3 station using the Hardware Manager.<br>For more information, refer to chapter **Hardware Manager** (see page 272). |
| **9** | Enter the sample program. Then, upload the program to the controller. |

# Configuration in JetSym

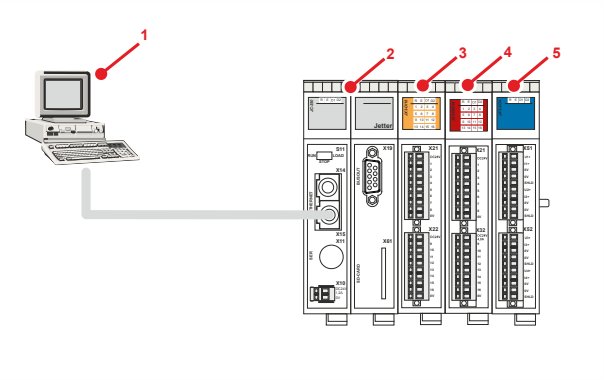
**Introduction**

A simple example is to illustrate configuration in JetSym: Connect the peripheral module JX3-DIO16 as a second module to a JC-3xx controller.

In a minimum program, a flashing light has been programmed. Output 9 of the module JX3-DIO16 is set and reset again.

**Configuration**

This example is based on the following configuration:




| Number | Part | Description |
|---|---|---|
| **1** | PC | Programming system |
| **2** | JC-3xx | Controller |
| **3** | JX3-DI16 | Peripheral module |
| **4** | JX3-DIO16 | Peripheral module |
| **5** | JX3-AO4 | Peripheral module |

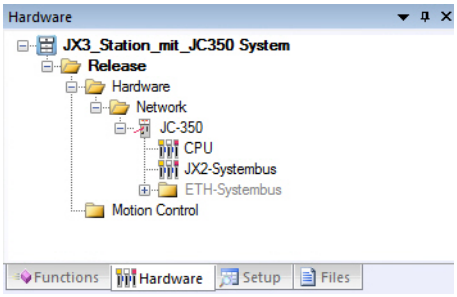
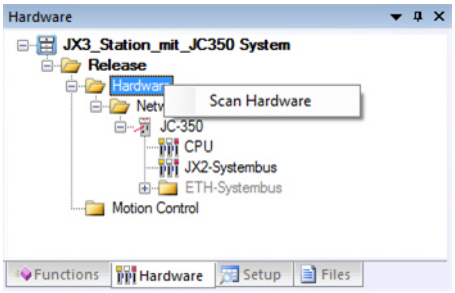
**Important note!**

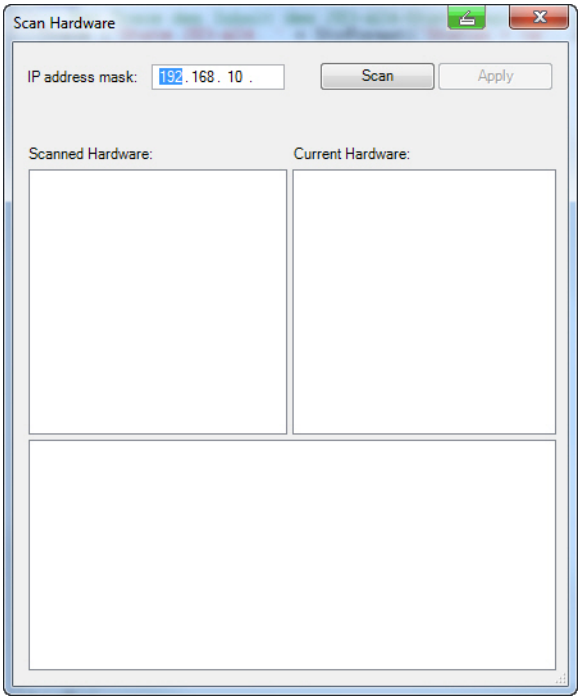
Connect the power supply to the terminals X21.DC24V/X21.0V and X32.DC24V/X32.0V of the module JX3-DIO16. Now, you can activate the digital outputs X32.9...16.

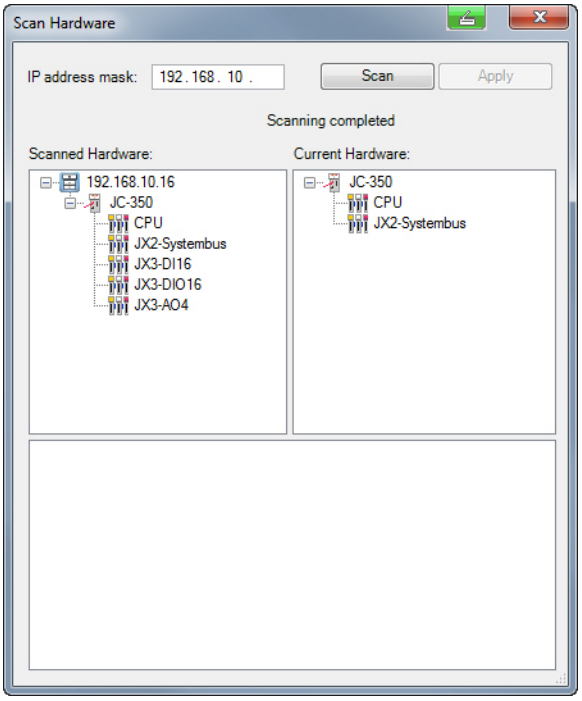
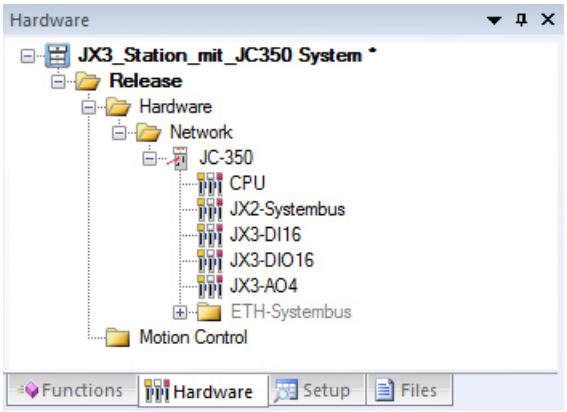
**Preparatory work**

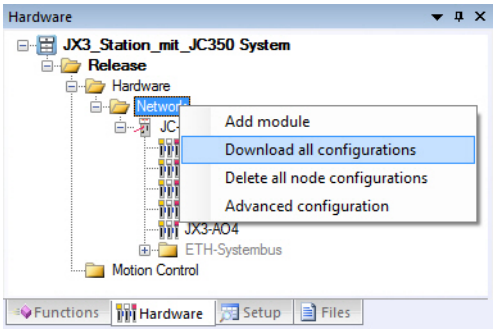
To properly configure the JX3 station, proceed as follows:

| Step | Action |
|---|---|
| **1** | Start the programming tool JetSym. |
| **2** | Create a new project. |

| Step | Action |
|---|---|
| **3** | In JetSym, start the Hardware Manager by clicking on the tab **Hardware** with the mouse or by pressing **Alt + 5** on your keyboard.<br>**Result**:<br>  |
| **4** | Connect the following Jetter AG products with the PC via Ethernet system bus:<br>■ Controller<br>■ Peripheral modules |
| **5** | Switch on the power supply. |
| **6** | In Hardware Manager, click on the folder **Hardware**. Right-click **Scan Hardware**.<br>  |

| Step | Action |
|---|---|
| **7** | Enter an IP address mask.<br>A hardware scan applies to the whole IP subnet. Therefore, enter at least the first three elements of the IP address. In this example, the IP address of the controller JC-3xx is 192.168.10.16.<br>To detect the control systems and all bus nodes, enter 192.168.10.<br>  |
| **8** | Click the button **Scan**. |

| Step | Action |
|---|---|
| ⇨ | The Hardware Manager scans the Jetter Ethernet system bus and compares the scanned hardware with the currently set hardware.<br>  |
| **9** | In the window **Scanned Hardware**, click the name of the controller. In this example, it is JC-350. |
| ⇨ | The Hardware Manager has the tree of the controller JC-350 displayed in the bottom window. |
| **10** | Click the button **Apply.** |
| ⇨ | The window closes. The Hardware Manager has taken over the hardware parameters.<br>  |
| **11** | Check the result of the automatic hardware scan. |

| Step | Action |
|---|---|
| **12** | Upload the given configuration to the controller.<br>  |
| **13** | Activate the programming environment by entering **Alt + 0** on your keyboard. As an alternative, you can click the tab **File**. |
| **14** | Enter the program shown below. |
| **15** | Upload the program to the controller. |
| ⇨ | LED **9** of the peripheral module JX3-DIO16 will be flashing.<br>This output will be refreshed every 5 seconds. |

**JetSym STX program**

Output 9 of the module JX3-DIO16 is set and then reset again.

```
Task Flashing_light Autorun
    Loop
        OUTPUTS[100000309] := True;
        Delay(T#5s);
        OUTPUTS[100000309] := False;
        Delay(T#5s);
    End_Loop;
End_Task;
```

**Related topics**

- **Hardware Manager** (see page 272)

# 6 File system

**Introduction**

This chapter describes the file system of the JC-360. The file system lets you access files located on the internal flash disk and the SD card.

**File categories**

The files of the file system are categorized as follows:

- System directories or system files used by the operating system
- Files which are at the user's disposal

**System directories**

The system directories cannot be deleted. System directories even survive formatting.

| Directory | Description |
|---|---|
| /System | ■ System configuration<br>■ System information |
| /SD | ■ Root directory of the SD memory card |

**Inhalt**


| Topic | Page |
|---|---|
| Properties | 154 |
| User administration | 157 |
| Reviewing the flash disk capacity used | 166 |
| Operating system update and application program | 170 |
| Formatting and checking | 171 |

# 6.1 Properties

**Introduction**

This chapter covers the properties of the file system. The description distinguishes between the internal flash disk and the SD card.

**General properties**

The following properties apply to the internal flash disk and the SD card:

- 8 files max. to be opened simultaneously
- Separate directory names by a slash "/", not by a backslash "\".
- When the controller creates a file, the file contains date and time assigned by the controller.
- Date, time, and file size are not available for all system files.

**Contents**


| Topic | Page |
|---|---|
| Flash disk - Properties | 155 |
| SD card - Properties | 156 |

# Flash disk - Properties

**Capacity**

The following disk space is available to the user:

| Parameter | Value |
|---|---|
| Flash disk capacity | 4 MB |

**Properties**

The internal flash disk drive has got the following properties:

- Up to 7 directory levels and 1 file level are allowed.
- Differentiation between upper and lower case.
- Directory and file names with a length of up to 63 characters are possible.
- All characters except "/" and ".." are permitted for directory and file names
- User/access administration for a maximum number of 31 locks and 33 users.

## SD card - Properties

**Capacity**

The available capacity depends on the SD card used:

| Parameter | Value |
|---|---|
| Tested capacities | 8 MB ... 4 GB |

**Properties**

The SD memory card has got the following properties:

- The SD memory card must be compatible with FAT 16.
- Directory and file names of 260 characters' length max. can be used.
- Differentiation between upper and lower case.
- The following characters are not permitted in directory and file names: "/", "\", ":", "*", "?", "<", ">" and "|"
- There is no user/access administration.

# 6.2 User administration

**Introduction**

The file system for the internal flash disk lets you define authorization for access (locks) to directories, and set up users.

For each user, you can set individual access rights (keys).

Users are not allowed to access directories and files for which they do not have the required key. In case of an FTP/IP connection, these directories and files are not displayed.

**Prerequisites**

Administrator rights are required for user administration.

**Properties**

The properties of user administration are as follows:

| Property | Max. value |
|---|---|
| Number of users | 33 |
| Number of predefined users | 2 |
| Length of a user name | 31 alphanumeric characters |
| Password length | 31 alphanumeric characters |
| Number of keys for read access | 31 |
| Number of keys for write access | 31 |
| Number of predefined keys | 2 |

**Files**

You can make settings for user administration in three files located in the directory **System**:

| File | Function |
|---|---|
| flashdisklock.ini | Assignment of locks to directories |
| keys.ini | Assignment of names to locks/keys |
| users.ini | Administration of users |

These files are always existing. They cannot be deleted, but only modified or overwritten.

**Restrictions**

Please take the following restrictions into account:

- User administration can only be applied to the internal flash disk. It cannot be applied to SD cards.
- If user administration has been assigned to a file, its contents are readable at once. The settings become active only after a reboot.

**Contents**


| Topic | Page |
|---|---|
| Administration of users | 159 |
| As-delivered condition/Predefined users and keys | 161 |
| Assigning locks | 162 |
| Assigning names to keys/locks | 164 |

# Administration of users

**Introduction**

The configuration file **/System/users.ini** lets you manage the user administration for the file system.

**Prerequisites**

If you want to use names for the keys, you must make them known to the device beforehand. Therefore, set up the names first as described in *Setting up names for keys/locks* (see page 164).

**Administration of users**

To manage user administration, proceed as follows:

| Step | Action |
|---|---|
| **1** | Establish an FTP connection to the device. Log on as administrator. |
| **2** | Open the file **/System/users.ini**. |
| **3** | Enter the required information. |
| **4** | Save the changed file to the device. |
| **5** | Reboot the device. |

**Result:** The changed user administration settings are now enabled.

**Structure of the configuration file**

This configuration file is a text file the entries of which are grouped into several sections.

- For each user a separate section is to be created.
- In these sections values can be set which are then used by the file system.
- You can insert blank lines as required.
- The following characters precede a comment line: "!", "#" or ";".

**Sections**

The sections are named *[USER1]* through *[USER33]*. Here, the user name and the related password, as well as read and write permissions are specified.

**Example:**

```
[USER4]
NAME=TestUser3
PW=testpass
READKEYS=5,openLock2,10,11
WRITEKEYS=openLock2,10,11
SYSKEYS=
```

| | |
|---|---|
| **NAME** | |
| In the given example | TestUser3 |
| Description | User's login name |
| Allowed values | A maximum of 31 alphanumeric characters |
| In case of illegal value or missing entry | User administration settings are not made |
| **PW** | |
| In the given example | testpass |
| Description | User's login password |
| Allowed values | A maximum of 31 alphanumeric characters |
| In case of missing entry | The user is allowed to log in without password |
| **READKEYS** | |
| In the given example | 5,openLock2,10,11 |
| Description | Keys for read access (read keys) |
| Allowed values | 1 ... 31 (or corresponding names) |
| In case of missing entry | No read keys are assigned to the user |
| **WRITEKEYS** | |
| In the given example | openLock2,10,11 |
| Description | Keys for write access (write keys) |
| Allowed values | 1 ... 31 (or corresponding names) |
| In case of missing entry | No write keys are assigned to the user |
| **SYSKEYS** | |
| Description | No function assigned; reserved for future extensions |

# As-delivered condition/Predefined users and keys

**Introduction**

Two predefined users with set rights are included in the file system. It is not possible to delete these two users. The user administration lets you only change the password for these two users.

**Factory settings**

In delivered condition the content of the configuration file included in the controller is as follows.

```
[USER1]
NAME=admin
PW=admin
READKEYS=1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31
WRITEKEYS=1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31
SYSKEYS=

[USER33]
NAME=system
PW=system
READKEYS=2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31
WRITEKEYS=2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31
SYSKEYS=
```

**Predefined users**

All keys are available to this user *admin*, and he/she is, therefore, able to read all directories and files and to write to them.

All keys except for key **1** are available to user *system*, too.

**Predefined keys**

Two out of the 31 keys have a predefined function.

| Lock/key | Function |
|---|---|
| 1 | ■ Ethernet configuration<br>■ User administration |
| 2 | ■ Operating system update of the CPU<br>■ Operating system update of JX2 and JX3 modules |

# Assigning locks

**Introduction**

In the configuration file **/System/flashdisklock.ini** you assign locks to directories located on the flash disk. Only users with the corresponding key are allowed to read or write (delete) files and subdirectories located in these directories.

**Prerequisites**

If you want to use names for the locks, you must make them known to the device beforehand. Therefore, set up the names first *Setting up names for keys/locks* (see page 164).

**Installing the lock**

To assign a lock to a directory, proceed as follows:

| Step | Action |
|---|---|
| **1** | Establish an FTP connection to the device; when doing so, log in with administrator rights. |
| **2** | Open the file **/System/flashdisklock.ini**. |
| **3** | Adjust the file entries. |
| **4** | Save the changed file to the device. |
| **5** | Reboot the device. |

**Result:** A lock is assigned to this directory.

**Structure of the configuration file**

This configuration file is a text file containing one section.

- In this section values can be set which are then used by the file system.
- Specify each directory with its lock number in an individual line.
- You can insert blank lines as required.
- The following characters precede a comment line: "!", "#" or ";".

**Section**

The section is named *[LOCKS]*. Here, locks are assigned to directories in accordance with the following rule:

Directory=Lock

**Example:**

```
[LOCKS]
test1=0
test1/sub1=2
test1/sub2=5
test2=userlock2
```

**Lock numbers**

Use the following lock numbers:

- The valid lock numbers are 0 ... 31.
- Lock number 0: No lock is assigned to this directory. You can access this directory without any special permissions.
- You can use numbers or previously defined names.

# Assigning names to keys/locks

**Introduction**

Keys/locks are consecutively numbered from 1 through 31. To provide ease of handling, a name can be assigned to each key/lock combination. These names are assigned in the configuration file **/System/keys.ini**.

**Configuring names**

To assign names to keys/locks, proceed as follows:

| Step | Action |
|---|---|
| **1** | Establish an FTP connection to the device; when doing so, log in with administrator rights. |
| **2** | Open the file **/System/keys.ini**. |
| **3** | Adjust the file entries. |
| **4** | Save the changed file to the device. |
| **5** | Reboot the device. |

**Result:**

The names are available now. The names are now available and can be used when assigning locks and managing user accounts.

**Structure of the configuration file**

This configuration file is a text file containing one section.

- In this section values can be set which are then used by the file system.
- Each key is specified with its name in an individual line.
- You can insert blank lines as required.
- The following characters precede a comment line: "!", "#" or ";".

**Section**

The section is named *[KEYS]*. Here, names are assigned to keys/locks in accordance with the following rule:

KEYxx=Name

xx: Number of the key (01 ... 31)

**Example:**

```
[KEYS]
KEY01=Admin
KEY02=System
KEY03=
KEY04=
KEY05=service
...
KEY31=
```

**Names for Locks/Keys**

For names the following definitions are true:

- A maximum of 15 alphanumeric characters
- Lock and key have the same name.

# 6.3 Reviewing the flash disk capacity used

**Introduction**

You can view the application scope of the internal flash disk.
Details on the allocation of the application scope are given in this chapter.

**Contents**


| Topic | Page |
|---|---|
| Flash disk capacity used | 167 |

# Flash disk capacity used

**README**

You can view the application data area of the internal flash disk.

You can see the capacity used of the application data area from the file **/System/flashdiskinfo.txt**.

**Example**

In this example, the fictive capacity used of a flash disk in a JetControl 340 (4 MB) is shown:

```
Name  : flash disk
Date  : 25.11.2008
Time  : 15:04
Tracks: 64

Track   0:  sectors: 128  (used:  81 / blocked:  47 / free:   0)
Track   1:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track   2:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track   3:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track   4:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track   5:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track   6:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track   7:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track   8:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track   9:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  10:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  11:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  12:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  13:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  14:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  15:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  16:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  17:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  18:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  19:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  20:  sectors: 128  (used:  64 / blocked:  64 / free:   0)
Track  21:  sectors: 128  (used:  85 / blocked:  43 / free:   0)
Track  22:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  23:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  24:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  25:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  26:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  27:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  28:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  29:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  30:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  31:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  32:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  33:  sectors: 128  (used: 105 / blocked:   0 / free:  23)
Track  34:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
```

```
Track  35:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  36:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  37:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  38:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  39:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  40:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  41:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  42:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  43:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  44:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  45:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  46:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  47:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  48:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  49:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  50:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  51:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  52:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  53:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  54:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  55:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  56:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  57:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  58:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  59:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  60:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  61:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  62:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  63:  sectors: 128  (used:   0 / blocked:   0 / free: 128)

Total:  sectors: 8192  (used: 4175 / blocked: 154 / free: 3863)

Used   :  2120900 byte
Blocked:    78232 byte
Free   :  1962404 byte
Total  :  4161536 byte
```

**Elements of info file**

Tracks and sectors represent the administration units of the flash disk. The info file comprises the following elements:

| Element | Description |
|---|---|
| `Name` | Dedicated name of the flash disk |
| `Date/Time` | Point in time when the flash disk was formatted last |
| `Tracks` | Total number of tracks |
| `Track xx: sectors: 128` | Assignment of sectors of a track |
| `Total: sectors:` | Overall statistical data of the sectors |
| `Used` | Total number of used bytes |
| `Blocked` | Total number of blocked bytes |
| `Free` | Total number of available bytes |
| `Total` | Total size of the flash disk |

**States of the sectors**

The smallest administrative unit of the flash disk, i.e. the sector, may assume the following states:

| State | Description |
|---|---|
| `Used` | The sector is occupied by data. |
| `Blocked` | The sector is no longer occupied, but can not yet be used due to administrative reasons. |
| `Free` | The sector is not occupied and can be used. |

# 6.4 Operating system update and application program

**Introduction**

An OS update for a controller, an HMI or an I/O module, as well as access to the application program can be carried out via file system. For a detailed description on this topic refer to the following chapter:

- *Operating system update* (see page 605)
- *Application program* (see page 617)

# 6.5 Formatting and checking

**Introduction**

This chapter covers the following topics:

- Formatting the flash disk
- Formatting the SD card
- Checking the SD card

The internal flash disk needs not be checked using a separate function, since it provides maximum safety of its administrative structures by design.

**Operating principle**

When the device boots up, the OS system checks the content of the control register. The control register is part of the file system.

Depending on the value contained in this register the following functions are carried out:

- Formatting the flash disk
- Formatting the SD card
- Checking the SD card

**Register number**

The register number of the control register is dependent on the device:

| Device | Register number |
|---|---|
| JC-360 | 202936 |

**Contents**


| Topic | Page |
|---|---|
| Formatting the flash disk | 172 |
| Formatting the SD card | 173 |
| Checking the SD card | 174 |

## Formatting the flash disk

**Introduction**

In the following cases, reformatting the flash disk is required:

- When you upload an OS version that has got another flash disk format
- When information for flash disk administration has been destroyed

**Consequences**

- All files and directories located in the user area will be deleted!
- Formatting will not affect system files and directories.

**Formatting the flash disk**

To have the device format the internal flash disk, proceed as follows:

| Step | Action |
|---|---|
| **1** | Switch the device ON. |
| **2** | Enter value -999720373 (0xc4697a4b) into the control register 202936 of the file system. |
| **3** | Switch the device OFF. |
| **4** | Switch the device ON. |

**Result:** During the boot process of the JC-360, the flash disk is formatted and the control register 202936 is set to **0**.

# Formatting the SD card

**Introduction**

In the following cases, reformatting the SD card is required:

- When information for SD card administration has been destroyed

**Consequences**

All files and directories on the SD card will be deleted!

**Formatting the SD card**

To have the device format the SD card, proceed as follows:

| Step | Action |
|---|---|
| **1** | Switch the device ON. |
| **2** | Enter value -748362163 (0xd364e64d) into the control register 202936 of the file system. |
| **3** | Switch the device OFF. |
| **4** | Switch the device ON. |

**Result:** During the boot process of the JC-360, the SD card is formatted and the control register 202936 is set to **0**.

## Checking the SD card

**Introduction**

In the following cases, checking the SD card for faults is required:

- When the device was switched off during access to the SD card

**Consequences**

- All files and directories on the SD card will be checked and errors, if any, will be fixed.
  Following such a check, the administrative structures on the SD card will be in consistent condition.
- Depending on the SD card size and the number of files and directories the boot process duration of the JC-360 may extend to several minutes.

**Checking the SD card**

To have the device check the SD card, proceed as follows:

| Step | Action |
|---|---|
| **1** | Switch the device ON. |
| **2** | Enter value 748371092 (0x2c9b3c94) into the control register 202936 of the file system. |
| **3** | Switch the device OFF. |
| **4** | Switch the device ON. |

**Result:** While booting, the device checks the SD card. The value in the control register remains unchanged so that the card is checked whenever the JC-360 is rebooted.

**Restrictions**

This function **repairs** the administrative structures on the SD card in order that it can be used further. However, it may happen that the device cannot restore in all cases data of a file, which, for example, has been written incompletely.

# 7 FTP server

**Introduction**

The FTP server lets you handle directories and files using an FTP client. The files can be stored to the following storage media:

- Internal flash disk
- SD card

This chapter covers the login process and describes the commands supported by the FTP server.

**FTP clients**

The user has the option of using a command line FTP client, which comes with many PC operating systems, or graphic FTP tools.

**Number of possible connections**

The FTP server on the JC-360 is able to manage up to four FTP connections simultaneously.

Any additional FTP client, which tries to connect with the FTP server, will get no response to its request for establishing a connection.

**Supported commands**

The FTP server supports standardized commands. For more information refer to:

- FTP server help menu; connect with FTP server and enter the command *help* or *help binary*.
- In the Web, search for FTP and commands

If you do not wish to care about commands, we recommend using an FTP program, such as TotalCommander.

**Required programmer's skills**

To be able to use the functions described in this chapter, the programmer must be familiar with the following subjects:

- The user must be familiar with the file system of the JC-360
- The user must be familiar with IP networks
- The user must be familiar with commands

**Contents**


| Topic | Page |
|---|---|
| Logon | 176 |
| Example: Windows FTP client | 177 |

## Logon

**Logon**

To be able to access the file system via FTP, the FTP client must log on when the connection is established.

- As **Server Name** enter the IP address of the device.
- As **User Name** enter your user name, e.g. *admin*.
- As **Password** enter your password, e.g. *admin*.

**Factory settings**

The factory settings include two user accounts:

[USER1]
NAME=admin
PW=admin

[USER33]
NAME=system
PW=system

**Administration of users**

The user administration of the file system lets you change the passwords and add new users.

**Related topics**

- **User administration** (see page 157)

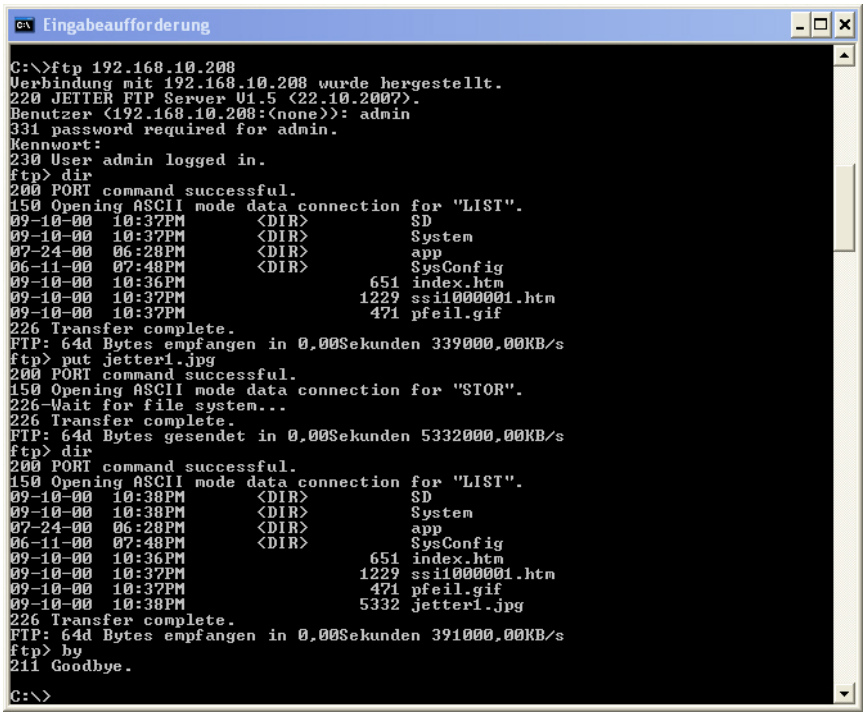
## Example: Windows FTP client

**Task**

Carry out the following tasks using an FTP client, for example, the one which comes with Windows:

- Launch the FTP client by opening a connection and entering the IP address.
- Log in as user *admin* with password *admin*
- Displaying the content of the current directory using *dir*
- Transferring the file **jetter1.jpg** to the JetControl using the command *put*
- Displaying the content of the current directory using *dir*
- Terminating the session and the FTP client using *bye*

**Action**



# 8 FTP client

**The FTP client**

The FTP client allows access from within the application program to files and directories of a remote network device. To this end, the FTP client communicates with the FTP server of this network device.

**Functions**

The following functions are possible:

- Creating directories in the remote file system.
- Deleting directories in the remote file system.
- Copying files from the local file system into the remote file system.
- Copying files from the remote file system into the local file system.

**Requirements**

- To be able to use the FTP client feature basic knowledge of FTP connections and file systems is required.
- The IP address of the FTP server must be known.
- If the IP address of the FTP server is not known, name resolution through a DNS server must be possible.
- User name and password for logging on at the FTP server must be known.
- For programming this feature JetSym version 4.3 or higher is required.

**Processing within the application program**

- The controller completes only one FTP access at a time.
- The corresponding task in the application program stops at the command until the access is completed.
- During this time other tasks in the application program are processed.
- While an FTP access of a task is being processed, all other tasks which invoke an FTP command are blocked until the FTP access is completed.

**Contents**


| Topic | Page |
|---|---|
| Programming | 180 |
| Registers | 199 |

# 8.1 Programming

**Introduction**

The FTP client allows to access files and directories on a network device from within the application program. For this purpose, function calls are used. These function calls are included in the programming language of the controller. To program this feature, proceed as follows:

| Step | Action |
|---|---|
| 1 | Initialize the FTP client |
| 2 | Open the connections to the FTP servers |
| 3 | Transfer data |
| 4 | Terminate the connections |

**Restrictions**

While the controller is processing one of the functions of the FTP client, tasks supporting the FTP client should not be stopped through `TaskBreak` or restarted through `TaskRestart`. Otherwise the controller fails to complete this function which will block new function calls by the FTP client.

**Contents**


| Topic | Page |
|---|---|
| Initializing the FTP client | 181 |
| Establishing a connection to the FTP server | 182 |
| Terminating a connection | 184 |
| Reading a file | 185 |
| Writing to a file | 187 |
| Deleting a file | 189 |
| Changing directories | 191 |
| Creating a directory | 193 |
| Deleting directories | 195 |
| Determining the current directory | 197 |

# Initializing the FTP client

**Introduction**

At each application program start, the FTP client must be initialized at least once.

**Function declaration**

```
Function FtpInitialize():Int;
```

**Return value**

The following return value is possible:

| Return value | |
|---|---|
| 0 | Always |

**How to apply this function**

The function is used and its return value assigned to a variable for further utilization in the following way:

```
Result := FtpInitialize();
```

**Operating principle**

The controller processes this function in the following steps:

| Step | Description |
|---|---|
| **1** | The controller closes all open connections of the FTP client. |
| **2** | The controller initializes all OS-internal data structures of the FTP client. |

## Establishing a connection to the FTP server

**Introduction**

Before data can be sent or received, a connection to the FTP server must be established first. When establishing the connection, the client logs in to the FTP server by a user name and a password (login).

**Function declaration**

```
Function FtpConnect(Const Ref ServerAddr: String,
                    Const Ref UserName: String,
                    Const Ref PassWord: String):Handle;
```

**Function parameters**

Description of the function parameters:

| Parameter | Value | Remarks |
|---|---|---|
| ServerAddr | IP address or name | Name resolution by DNS server |
| UserName | User name for logon | Login |
| Password | Password for logon | Login |

**Return value**

If the return value was positive, the connection could be established and login was successful. If the return value was **0**, an error occurred and the connection could not be established.

| Return value | |
|---|---|
| > 0 | A positive return value must be stored in a variable. It must be made available as a handle at activating the following functions:<br>■ `Terminating a connection`<br>■ `Reading a file from the FTP server`<br>■ `Writing a file to the FTP server`<br>■ `Deleting a file from the FTP server`<br>■ `Changing a directory on the FTP server`<br>■ `Creating a directory on the FTP server`<br>■ `Deleting a directory from the FTP server`<br>■ `Determining the current directory on the FTP server` |
| 0 | Error when establishing a connection or logging in to the FTP server |

**Operating principle**

The task stops at the program line until the connection is established or the timeout set for the FTP client has elapsed.

This function is processed in the following steps:

| Step | Description | |
|---|---|---|
| **1** | The controller tries to establish a TCP/IP connection to the FTP server. | |
| **2** | **If ...** | **... then ...** |
| | ... the network client has accepted the connection, ... | ... proceed with step 3. |
| | ... the connection could not be established and the timeout has not elapsed yet, ... | ... proceed with step 1. |
| | ... an error has occurred or the timeout has elapsed, ... | ... the function is terminated and value **0** is returned. |
| **3** | The controller logs on to the FTP server with its user name *Administrator* and password *AdminPassword*. | |
| **4** | **If ...** | **... then ...** |
| | ... the FTP server has accepted the connection, ... | ... the function is terminated and a positive value is returned as handle for further access to this connection. |
| | ... the FTP server has not accepted the connection, for example because of an invalid user name or wrong password, ... | ... the function is terminated and value **0** is returned. |

**Related topics**

- **Terminating a connection** (see page 184)

## Terminating a connection

**Introduction**

Clear all connections which are no longer required as this will reduce PLC load for managing connections.

**Function declaration**

```
Function FtpDisconnect(FtpConnection:Handle):Int;
```

**Function parameters**

Description of the function parameters:

| Parameter | Value | Remarks |
|---|---|---|
| FtpConnection | Handle | Value returned by the function `FtpConnect()` |

**Return value**

The following return values are possible:

| Return value | |
|---|---|
| 0 | Connection terminated and deleted |
| -1 | Invalid handle |
| -2, -3 | Communication error, there is, for example, no response from FTP server |

**Related topics**

- **Establishing a connection to the FTP server** (see page 182)

# Reading a file

**Introduction**

This function lets you read the content of a file from a remote network node and copy it to the local file system of the controller.

**Function declaration**

```
Function FtpFileRead(FtpConnection:Handle,
                     Const Ref ServerFile: String,
                     Const Ref ClientFile: String):Int;
```

**Function parameters**

Description of the function parameters:

| Parameter | Value | Remarks |
|---|---|---|
| FtpConnection | Handle | Value returned by the function `FtpConnect()` |
| ServerFile | File name | Name of the file in the file system of the FTP server, which the controller is to read |
| ClientFile | File name | File name, as which the controller is to save the file read in the local file system |

**Return value**

If the returned value is negative, an error has occurred. If the returned value is **0**, the controller was able to read the file and store it locally.

| Return value | |
|---|---|
| 0 | No error |
| -1 | Invalid handle |
| -2, -6 | Error when storing the file locally |
| -3, -5, -7, -8 | Communication error, there is, for example, no response from FTP server |
| -4 | Error message from FTP server, for example, file does not exist |

**Operating principle**

In the following cases, the task is not processed further after issuing the function call:

- The controller must first read the file, e. g. **ServerTestFile.txt** and save it to the local file system as, e. g., **LocalTestFile.txt**.
- An error has occurred.

This function is processed in the following steps:

| Step | Description | |
|---|---|---|
| 1 | The controller sends a command to the FTP server that the content of the file **ServerTestFile.txt** must be transmitted. | |
| 2 | The controller receives the contents of the file **ServerTestFile.txt**. | |
| 3 | The controller writes the contents to the file **LocalTestFile.txt**. | |
| 4 | **If ...** | **... then ...** |
| | ... no errors have occurred, ... | ... the file has been copied successfully, the function is terminated and value **0** is returned. |
| | ... errors have occurred, ... | the function is terminated and a negative value is returned. |

**File names**

- The function parameter for the local file may contain the path to this file, e.g. `"/Data/TestFiles/LocalTestFile.txt"`.
- If the file system of the remote network node supports this, the function parameter for the file located on the FTP server can also contain the path to this file. Otherwise, the directory must be set beforehand using the command `FtpDirChange()`.
- The file system of a JC-360 supports both options.

**Related topics**

- **Writing to a file** (see page 187)

## Writing to a file

**Introduction**

This function lets you copy the content of a file belonging a local file system to a file belonging to the file system of a remote network node.

**Function declaration**

```
Function FtpFileWrite(FtpConnection:Handle,
                      Const Ref ServerFile: String,
                      Const Ref ClientFile: String):Int;
```

**Function parameters**

Description of the function parameters:

| Parameter | Value | Remarks |
|---|---|---|
| FtpConnection | Handle | Value returned by the function `FtpConnect()` |
| ServerFile | File name | File name as which the FTP server is to save the written file |
| ClientFile | File name | Name of the file in the local file system, the content of which the controller is to send to the FTP server |

**Return value**

If the returned value is negative, an error has occurred. If the returned value is **0**, the controller was able to read the file and store it to the file system of the remote network node.

| Return value | |
|---|---|
| 0 | No error |
| -1 | Invalid handle |
| -2 | Error when reading the local file, e.g. file does not exist |
| -3, -5, -8 | Communication error, there is, for example, no response from FTP server |
| -4, -7 | Error message from the FTP server, e.g. file cannot be created |

**Operating principle**

In the following cases, the task is not processed further after issuing the function call:

- The controller must first read the file, e. g. **LocalTestFile.txt** and save it to the file system of the remote network node as, e. g., **ServerTestFile.txt**.
- An error has occurred.

This function is processed in the following steps:

| Step | Description | |
|---|---|---|
| **1** | The controller sends a command to the FTP server that the content of the file **ServerTestFile.txt** must be saved. | |
| **2** | The controller sends the contents of the file **LocalTestFile.txt**. | |
| **3** | The FTP server writes the contents to the file **ServerTestFile.txt**. | |
| **4** | **If ...** | **... then ...** |
| | ... no errors have occurred, ... | ... the file has been copied successfully, the function is terminated and value **0** is returned. |
| | ... errors have occurred, ... | the function is terminated and a negative value is returned. |

**File names**

- The function parameter for the local file may contain the path to this file, e.g. `"/Data/TestFiles/LocalTestFile.txt"`.
- If the file system of the remote network node supports this, the function parameter for the file located on the FTP server can also contain the path to this file. Otherwise, the directory must be set beforehand using the command `FtpDirChange()`.
- The file system of a JC-360 supports both options.

**Related topics**

- **Reading a file** (see page 185)

# Deleting a file

**Introduction**

This function lets you remove a file from the file system of a remote network node.

**Function declaration**

```
Function FtpFileRemove(FtpConnection:Handle,
                       Const Ref ServerFile: String):Int;
```

**Function parameters**

Description of the function parameters:

| Parameter | Value | Remarks |
|---|---|---|
| FtpConnection | Handle | Value returned by the function `FtpConnect()` |
| ServerFile | File name | Name of the file to be removed. |

**Return value**

If the returned value is negative, an error has occurred. If the returned value is **0**, the file could not be removed from the file system of the remote network node.

| Return value | |
|---|---|
| 0 | No error |
| -1 | Invalid handle |
| -2 | Communication error, there is, for example, no response from FTP server |
| -3 | Error message from FTP server, for example, file does not exist |

**Operating principle**

In the following cases, the task is not processed further after issuing the function call:

- The FTP server must first delete the file **ServerTestFile.txt**. Please note: This file name serves as an example only.
- An error has occurred.

This function is processed in the following steps:

| Step | Description | |
|---|---|---|
| **1** | The controller sends a command to the FTP server that the file **ServerTestFile.txt** must be deleted. | |
| **2** | The FTP server deletes the file **ServerTestFile.txt**. | |
| **3** | **If ...** | **... then ...** |
| | ... no errors have occurred, ... | ... the file has been deleted successfully, the function is terminated and value **0** is returned. |
| | ... errors have occurred, ... | the function is terminated and a negative value is returned. |

**File names**

- The function parameter for the local file may contain the path to this file, e.g. `"/Data/TestFiles/LocalTestFile.txt"`.
- If the file system of the remote network node supports this, the function parameter for the file located on the FTP server can also contain the path to this file. Otherwise, the directory must be set beforehand using the command `FtpDirChange()`.
- The file system of a JC-360 supports both options.

# Changing directories

**Introduction**

This function lets you remove the current directory from the file system of a remote network node.

**Function declaration**

```
Function FtpDirChange(FtpConnection:Handle,
                      Const Ref ServerDir: String):Int;
```

**Function parameters**

Description of the function parameters:

| Parameter | Value | Remarks |
|---|---|---|
| FtpConnection | Handle | Value returned by the function `FtpConnect()` |
| ServerDir | Directory name | Name of the directory into which the user wants to change |

**Return value**

If the returned value is negative, an error has occurred. If the returned value is **0**, the system has managed to change directories.

| Return value | |
|---|---|
| 0 | No error |
| -1 | Invalid handle |
| -2 | Communication error, there is, for example, no response from FTP server |
| -3 | Error message from the FTP server, e.g. directory does not exist |

**Operating principle**

In the following cases, the task is not processed further after issuing the function call:

- The FTP server must first change directories.
- An error has occurred.

This function is processed in the following steps:

| Step | Description | |
|---|---|---|
| **1** | The controller sends a command to the FTP server that it has to change to a subdirectory. | |
| **2** | The FTP server changes directories. | |
| **3** | **If ...** | **... then ...** |
| | ... no errors have occurred, ... | ... the new directory is set, the function is terminated and value **0** has been returned. |
| | ... errors have occurred, ... | the function is terminated and a negative value is returned. |

**Directory names**

- If the file system of the remote network node supports this, the function parameter for the directory located on the FTP server can also contain the complete path including several subdirectories leading to this directory.
- If this feature is not supported, the user must navigate from one directory level to the next until the desired directory is reached. This is done using the command `FtpDirChange()`.
- The file system of a JC-360 supports both options.

**Related topics**

- **Determining the current directory** (see page 197)

# Creating a directory

**Introduction**

This function lets you create a new directory from the file system of a remote network node.

**Function declaration**

```
Function FtpDirCreate(FtpConnection:Handle,
                      Const Ref ServerDir: String):Int;
```

**Function parameters**

Description of the function parameters:

| Parameter | Value | Remarks |
|---|---|---|
| FtpConnection | Handle | Value returned by the function `FtpConnect()` |
| ServerDir | Directory name | Name of the directory to be created |

**Return value**

If the returned value is negative, an error has occurred. If the returned value is **0**, the directory could successfully be created in the file system of the remote network node.

| Return value | |
|---|---|
| 0 | No error |
| -1 | Invalid handle |
| -2 | Communication error, there is, for example, no response from FTP server |
| -3 | Error message from FTP server, e.g. directory already exists |

**Operating principle**

In the following cases, the task is not processed further after issuing the function call:

- The FTP server must first create a subdirectory.
- An error has occurred.

This function is processed in the following steps:

| Step | Description | |
|---|---|---|
| **1** | The controller sends a command to the FTP server that it has to create a subdirectory. | |
| **2** | The FTP server creates the directory. | |
| **3** | **If ...** | **... then ...** |
| | ... no errors have occurred, ... | ... the new directory has been created, the function is terminated and value **0** is returned. |
| | ... errors have occurred, ... | ... the function is terminated and a negative value is returned. |

**Directory names**

- If the file system of the remote network node supports this, the function parameter for the directory located on the FTP server can also contain the complete path including several subdirectories leading to this directory.
- If this feature is not supported, the user must navigate from one directory level to the next until the desired directory is reached. This is done using the command `FtpDirChange()`.
- The file system of a JC-360 supports both options.

**Restrictions regarding the file system of a JetControl**

If you specify a directory with the corresponding path as function parameter, all directories up to the directory you want to create must exist. Recursive creation of several directories is not supported.

**Example:**

```
Result := FtpDirCreate(FtpHandle,
                       '/DataFiles/TextFiles/Release');
```

To be able to create the folder **Release** in the directory tree */DataFiles/TextFiles* the directories must already exist.

**Related topics**

- **Deleting directories** (see page 195)

# Deleting directories

**Introduction**

This function lets you remove a directory from the file system of a remote network node.

**Function declaration**

```
Function FtpDirRemove(FtpConnection:Handle,
                      Const Ref ServerDir: String):Int;
```

**Function parameters**

Description of the function parameters:

| Parameter | Value | Remarks |
|---|---|---|
| FtpConnection | Handle | Value returned by the function `FtpConnect()` |
| ServerDir | Directory name | Name of the directory to be deleted |

**Return value**

If the returned value is negative, an error has occurred. If the returned value is **0**, the directory could successfully be removed from the file system of the remote network node.

| Return value | |
|---|---|
| 0 | No error |
| -1 | Invalid handle |
| -2 | Communication error, there is, for example, no response from FTP server |
| -3 | Error message from the FTP server, e.g. directory does not exist |

**Operating principle**

In the following cases, the task is not processed further after issuing the function call:

- The FTP server must first remove the subdirectory.
- An error has occurred.

This function is processed in the following steps:

| Step | Description | |
|---|---|---|
| **1** | The controller sends a command to the FTP server that it has to remove the subdirectory. | |
| **2** | The FTP server removes the subdirectory. | |
| **3** | **If ...** | **... then ...** |
| | ... no errors have occurred, ... | ... the directory is removed, the function is terminated and value **0** is returned. |
| | ... errors have occurred, ... | the function is terminated and a negative value is returned. |

**Directory names**

- If the file system of the remote network node supports this, the function parameter for the directory located on the FTP server can also contain the complete path including several subdirectories leading to this directory.
- If this feature is not supported, the user must navigate from one directory level to the next until the desired directory is reached. This is done using the command `FtpDirChange()`.
- The file system of a JC-360 supports both options.

**Related topics**

- **Creating a directory** (see page 193)

# Determining the current directory

**Introduction**

This function lets you determine the current directory in the file system of a remote network node.

**Function declaration**

```
Function FtpDirPrint(FtpConnection:Handle,
                     Ref str: String):Int;
```

**Function parameters**

Description of the function parameters:

| Parameter | Value | Remarks |
|---|---|---|
| FtpConnection | Handle | Value returned by the function `FtpConnect()` |
| str | String address | Current directory with path specification |

**Return value**

If the returned value is negative, an error has occurred. If the returned value is **0**, the current directory could successfully be determined in the file system of the remote network node.

| Return value | |
|---|---|
| 0 | No error |
| -1 | Invalid handle |
| -3 | Communication error, there is, for example, no response from FTP server |
| -4 | Error message sent by the FTP server |
| -5 | Invalid response from server |

**Operating principle**

In the following cases, the task is not processed further after issuing the function call:

- The FTP server must first determine the actual directory.
- An error has occurred.

This function is processed in the following steps:

| Step | Description | |
|---|---|---|
| **1** | The controller sends a command to the FTP server that it has to determine the current directory. | |
| **2** | The FTP server transmits the actual directory with path specification. | |
| **3** | **If ...** | **... then ...** |
| | ... no errors have occurred, ... | ... the variable contains the complete path of the current directory, the function is terminated and value **0** is returned. |
| | ... errors have occurred, ... | the function is terminated and a negative value is returned. |

**Related topics**

- **Changing directories** (see page 191)

# 8.2 Registers

**Introduction**

This chapter describes the registers on the controller which contain status information of the FTP client. These registers can be used for debugging or diagnostic purposes. However, they can't be used for other functions such as establishing or terminating a connection.

**Contents**


| Topic | Page |
|---|---|
| Register numbers | 200 |
| Registers - Description | 201 |

## Register numbers

**Introduction**

Data of one connection each are displayed within the registers of a coherent register block. Two other registers show the status of the command being executed by the application program. The basic register number of these registers is dependent on the controller.

**Register numbers**

| Device | Basic register number | Register numbers |
|---|---|---|
| JC-360 | 320000 | 320000 ... 320101 |

**Determining the register number**

In this chapter, only the last three figures of a register number are specified. e.g. MR 002. To determine the complete register number, add to this module register number the basic register number of the corresponding device, for example 320000.

**Registers - Overview**

FTP client module registers - Overview

| Register | Description |
|---|---|
| **MR 000** | Number of open connections |
| **MR 002** | Timeout in seconds |
| **MR 003** | Port number of the FTP server |
| **MR 004** | Index in the connection table |
| **MR 005** | Connection handle |
| **MR 006** | IP address of the FTP server |
| **MR 007** | Port number of the FTP server |
| **MR 008** | IP address of FTP client |
| **MR 009** | Port number of FTP client |
| **MR 100** | Processing status on part of FTP client |
| **MR 101** | Task ID |

# Registers - Description

**Introduction**

Established connections are managed by the operating system of the controller in a list. Module registers MR 004 or 005 are used to copy connection data into registers MR 006 through MR 009.

**MR 000**

## Number of open connections

The value in this register shows how many connections are currently open.

| Module register properties | | |
|---|---|---|
| Reading values | 0 ... 2,147,483,647 | Number of connections |

**MR 002**

## Timeout

To this register, write the timeout of the FTP client at accessing the FTP server.

| Module register properties | | |
|---|---|---|
| Values | 0 ... 2,147,483,647 | in seconds |
| Value after reset | 20 | |

**MR 003**

## Port number of the FTP server

The value in this register shows the IP port number of the FTP server.

| Module register properties | |
|---|---|
| Values | 0 ... 65,535 |
| Value after reset | 21 |

**MR 004**

## Index in the connection table

The index of the connection table is entered into this register. If a connection has been established for a given index, the connection handle can be seen in module register MR 005 and connection data in module registers MR 006 through MR 009.

| Module register properties | |
|---|---|
| Values | 0 ... [MR 000] - 1 |
| Value after reset | -1 |

**MR 005**

## Connection handle

This register is for entering the connection handle. If a connection has been established for a given index, the connection handle can be seen in module register MR 004 and connection data in module registers MR 006 through MR 009.

| Module register properties | |
|---|---|
| Values | 0 ... 2,147,483,647 |

**MR 006**

## IP address of the FTP server

The value in this register shows the IP address of the FTP server.

| Module register properties | |
|---|---|
| Access | Read |
| Takes effect | if MR 004 >= 0 |

**MR 007**

## Port number of the FTP server

The value in this register shows the port number of the FTP server.

| Module register properties | |
|---|---|
| Access | Read |
| Takes effect | if MR 004 >= 0 |

**MR 008**

## IP address of FTP client

The value in this register shows the IP address of the FTP client.

| Module register properties | |
|---|---|
| Access | Read |
| Takes effect | if MR 004 >= 0 |

**MR 009**

## Port number of FTP client

The value in this register shows the port number of the FTP client.

| Module register properties | | |
|---|---|---|
| Access | Read | |
| Takes effect | if MR 004 >= 0 | |

**MR 100**

## Processing status on part of FTP client

This register lets you track the processing status on part of FTP client.

| Module register properties | | |
|---|---|---|
| Values | 0 | No access at the moment |
| | 1 | Parameters are being handed over to the FTP client of the controller |
| | 2 | The FTP client communicates with the FTP server. |
| | 3 | Access completed |
| Access | Read | |

**MR 101**

## Task ID

This register shows the ID of the task which is processing an FTP client function at that moment.

| Module register properties | | |
|---|---|---|
| Values | 0 ... 99 | Task ID |
| | 255 | None of the tasks is carrying out an FTP function. |
| Value after reset | 255 | |
| Access | Read | |

# 9 HTTP server

**Introduction**

A standard browser is sufficient for accessing the HTTP server.

The browser is for reading and displaying files which have been downloaded to the controller via FTP.

Here, it may be necessary to enter the user name and password to have access to certain pages (depending on the file system configuration).

**Default file names**

The default file names are **index.htm** and **index.html**.

**Supported file types**

The following file types are supported:

- *.htm, *.html, *.shtml
- *.txt, *.ini
- *.gif, *.tif, *.tiff, *.bmp, *.wbmp
- *.jpg, *.jpe, *.jpeg, *.png
- *.xml
- *.js, *.jar, *.java, *.class, *.cab
- *.ocx
- *.pdf, *.zip, *.doc, *.rtf
- *.css
- *.wml, *.wmlc, *.wmls, *.wmlsc
- *.ico, *.svg

**Enabling the HTTP server feature**

To enable the HTTP server feature in the controller the following requirements must be met:

- When ordering the controller option -W has been selected.

If this requirement has been met, bit 1 in Web status register 202930 is set.

In the controllers JC-360(MC) and JC-365(MC), the HTTP server feature is active by default.

**Required programmer's skills**

To be able to use the functions described in this chapter, the programmer must be familiar with the following subjects:

- The file system of the controller
- IP networks

**Contents**


| Topic | Page |
|---|---|
| Server Side Includes | 206 |

# 9.1 Server Side Includes

**Introduction**

The HTTP server features *Server Side Includes* (SSI). This function is for showing present real-time controller values on an HTML page.

**Rules**

You must specify a **Name Space** tag at the beginning of the HTML page that is to contain the real-time controller values.

This **Name Space** is for defining the namespace used in the HTML page.

In the body section of the HTML page the **Data** tags are specified.

**Updating real-time controller values**

When the HTML page is uploaded to the browser, the HTTP server once replaces the **Data** tags by actual real-time controller values.

To refresh the controller values, the HTML page must be reloaded over and over again.

The user triggers reloading by entering the controller address and the name of the required page, e.g. *http://192.168.10.209/Homepage/SSI/ssiTimeAndDate.htm*.

**Contents**


| Topic | Page |
|---|---|
| First entry in the HTML file | 207 |
| Inserting real-time controller values | 208 |
| Example of an HTML page | 213 |

## First entry in the HTML file

**Configuration**

The **Name Space** must be the first entry in the HTML file. It has got the following structure:

```
<NS:DTAG xmlns:NS=http://jetter.de/ssi/jetcontrol/
```

with **NS** representing the namespace. The namespace is a character string with a maximum length of 63 characters.

The namespace introduced here will be re-used for the subsequent **Data** tags. The remaining parts of the line are preassigned and have to be specified in exactly the same way.

In the following examples, the namespace applied is **JW**.

## Inserting real-time controller values

**Introduction**

Actual real-time controller values are integrated into parameter entries within the sections via tag functions. This way, the contents respectively states of registers, text registers, inputs, outputs and flags can be displayed.

**Tag delimiters**

All tags start and end with defined strings (delimiters). Between these tag delimiters, the variables are defined.

| Delimiter | String |
|---|---|
| Tag start | <JW:DTAG |
| Tag end | /> |

**Variable definition**

The variable definition in a tag contains attributes which are used to set, for example, how the value of a variable is to be displayed:

| **name** | |
|---|---|
| Function | Variable name |
| Comments | Code letter followed by the variable number |
| Example | `name="R1000023"` |

| **type** | |
|---|---|
| Description | Variable type of notation |
| Example | `type="REAL"` |

| **format** | |
|---|---|
| Description | Representation format |
| Comments | Refer to format definition |
| Example | `format="+0####.###"` |

| **factor** | |
|---|---|
| Description | Factor by which the real-time controller value is multiplied |
| Comments | Multiplication is executed prior to adding the offset |
| Example | `factor="1.5"` |

| **offset** | |
|---|---|
| Description | Value which is added to the real-time controller value |
| Comments | Multiplication by the factor is executed prior to adding the value to the real-time controller value |
| Example | `offset="1000"` |

**Format definition**

You can define the representation of variables by means of their attribute.

- The number of digits/characters used for representing a variable can be defined by the character "#".
- Prefix "0" sets the output of leading zeros. This applies to the register types INT, INTX and REAL.
- Prefix "+" sets the output of a sign. This applies to the register types INT and REAL.
- Prefixing a blank sets the output of a blank. This applies to the register types INT and REAL.

**Registers/text registers**

The variable name begins with a capital "R" followed by the register number.
The following types are possible:

| Type | Notation |
|---|---|
| INT (standard type) | Integer, decimal |
| INTX | Integer, hexadecimal |
| INTB | Integer, binary |
| BOOL | Register content = 0 --> Display: 0<br>Register content != 0 --> Display: 1 |
| REAL | Floating point, decimal |
| STRING | Text register |

**Example:**

```
JW:DTAG name="R1000250" type="REAL" format="+0####.###"
factor="3.25" offset="500" /
```

**Result:**

This instruction causes the contents of register 1000250 to be multiplied by 3.25. Then 500 is added to the product. The result appears in the Web browser with sign and at least five integer positions before the decimal point. Leading zeros are added as appropriate. Furthermore, three decimal positions are added.

**Flags**

The variable name begins with a capital "F" followed by the flag number.
The following types are possible:

| Type | Notation |
|---|---|
| BOOL (standard type) | Flag = 0 --> Display: 0<br>Flag = 1 --> Display: 1 |
| STRING | Flag = 0 --> Display: FALSE<br>Flag = 1 --> Display: TRUE |

**Example:**

```
<JW:DTAG name="F100" type="STRING" format="#" />
```

**Result:**
The state of flag 100 is displayed as string "T" or "F".

**Inputs**

The variable name begins with a capital "I" followed by the input number.
The following types are possible:

| Type | Notation |
|---|---|
| BOOL (standard type) | Input = 0 --> Display: 0<br>Input = 1 --> Display: 1 |
| STRING | Input = 0 --> Display: OFF<br>Input = 1 --> Display: ON |

**Example:**

```
<JW:DTAG name="I100000308" type="STRING" />
```

**Result:**
The state of input 100000308 is displayed as string "ON" or "OFF".

**Outputs**

The variable name begins with a capital "O" followed by the output number.
The following types are possible:

| Type | Notation |
|---|---|
| BOOL (standard type) | Output = 0 --> Display: 0<br>Output = 1 --> Display: 1 |
| STRING | Output = 0 --> Display: OFF<br>Output = 1 --> Display: ON |

**Example:**

```
<JW:DTAG name="O100000308" />
```

**Result:**
The state of output 100000308 is inserted as "1" or "0".

**Access via pointer register**

Access via pointer register is realized by inserting the capital letter "P" in front of the variable name. In each case the value of the variable is displayed whose number corresponds to the content of the register specified in the variable name.

**Examples:**

```
<JW:DTAG name="PR1000300" />
```

Result: The content of the register is displayed whose number is contained in register 1000300.

```
<JW:DTAG name="PF1000300" />
```

Result: The state of the flag is displayed whose number is contained in register 1000300.

```
<JW:DTAG name="PI1000300" />
```

Result: The state of the input is displayed whose number is contained in register 1000300.

```
<JW:DTAG name="PO1000300" />
```

Result: The state of the output is displayed whose number is contained in register 1000300.

**Access via pointer register and offset**

To specify the number of the variable to be displayed, it is also possible to add a constant value or another register content to the pointer register value

**Examples:**

```
<JW:DTAG name="PR1000300 + 100" />
```

**Result:** The content of the register is displayed whose number results from the addition of the content of register 1000300 and value 100.

```
<JW:DTAG name="PR1000300 + R1000100" />
```

**Result:** The content of the register is displayed whose number results from the addition of the content of register 1000300 and the content of register 1000100.

```
<JW:DTAG name="PF1000300 + 100" />
```

**Result:** The state of the flag is displayed whose number results from the addition of the content of register 1000300 and value 100.

```
<JW:DTAG name="PF1000300 + R1000100" />
```

**Result:** The state of the flag is displayed whose number results from the addition of the content of register 1000300 and the content of register 1000100.

```
<JW:DTAG name="PI1000300 + 100" />
```

**Result:** The state of the input is displayed whose number results from the addition of the content of register 1000300 and the value 100.

```
<JW:DTAG name="PI1000300 + R1000100" />
```

**Result:** The state of the input is displayed whose number results from the addition of the content of register 1000300 and the content of register 1000100.

```
<JW:DTAG name="PO1000300 + 100" />
```

**Result:** The state of the output is displayed whose number results from the addition of the content of register 1000300 and the value 100.

```
<JW:DTAG name="PO1000300 + R1000100" />
```

**Result:** The state of the output is displayed whose number results from the addition of the content of register 1000300 and the content of register 1000100.

## Example of an HTML page

**Task**

Insert current real-time controller values into an HTML page.

It should then be possible to display the HTML page in a browser using the *Server Side Includes* feature.

**Action**

```
<JC:DTAG xmlns:JC="http://jetter.de/ssi/jetcontrol" />
<html>

<head>
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1252">
<meta name="GENERATOR" content="Microsoft FrontPage 4.0">
<meta name="ProgID" content="FrontPage.Editor.Document">
<title>Index</title>
</head>

<body>
Hello World, 
<p>Actual controller values can be inserted into an html page like
this: </p>
<p>Register 201000 = <JC:DTAG name="R201000" type = INT
format="+####" />,
or Hex notation: 0x<JC:DTAG name="R201000" type="INTX"
format="0###" />,
or rather this way: <JC:DTAG name="R201000" type="BOOL" />, if only
Boolean is queried.
Binary notation is also an option: <JC:DTAG name="R201000"
type="INTB" format=######## />b. </p>
<p>Strings could also be defined "<JC:DTAG name="R201000"
type="STRING" />". </p>
<p>A real number looks as follows: <JC:DTAG name="R1001500"
type="REAL" />
or this way: <JC:DTAG name="R1001500" type="REAL" factor="1.3"
format="###.##" />. </p>
<p>The value of a flag is represented as follows: <JC:DTAG name="F10"
/>
or this way: <JC:DTAG name="F10" type="STRING" />. </p>
<p>With inputs and outputs, it is done the same way: <JC:DTAG
name="I100000205" type="BOOL" />
or <JC:DTAG name="I100000205" type="STRING" />. </p>
<p>R201000 = <JC:DTAG name="R201000" type="INT"
format="+0##########" /> </p>
<p>Regards </p>
<p>Yours JetControl</p>
</body>

</html>
```

**Storage location**

Now store the HTML page to the file system of the controller.

# 10 Programming

**Purpose of this chapter**

This chapter is for supporting you when programming a JC-360 controller in the following fields of activity:

- Determining the register numbers of connected modules
- Determining the I/O numbers of connected modules
- Programming additional functions
- Data interchange via various system buses
- Data interchange via user-programmable interfaces

**Prerequisites**

To be able to program the JC-360 the following prerequisites have to be fulfilled:

- The controller is connected to a PC.
- The JetSym programming software has been installed on the PC.

**Contents**


| Topic | Page |
|---|---|
| Abbreviations, module register properties and formats | 216 |
| Memories - Overview | 217 |
| Register and I/O numbers with a JC-3xx | 229 |
| Jetter Ethernet system bus | 240 |
| General system registers | 302 |
| Startup delay register | 311 |
| Real-time clock (RTC) | 313 |
| Runtime registers | 322 |
| Monitoring interface activities | 325 |
| Controlling HMIs with alphanumeric displays | 330 |
| Controlling printer and serial interfaces | 389 |
| JX2 system bus | 408 |
| JX3 system bus | 431 |
| E-mail | 441 |
| Sorting data | 465 |
| Modbus/TCP | 466 |
| User-programmable serial interface | 477 |
| User-programmable IP interface | 502 |
| CANopen® STX API | 521 |
| User-programmable CAN-Prim interface | 552 |

# Abbreviations, module register properties and formats

## Abbreviations

The abbreviations used in this document are listed in the table below:

| Abbreviation | Description |
|---|---|
| R 100 | Register 100 |
| MR 150 | Module register 150 |

## Module register properties

Each module register is characterized by certain properties. Most properties are identical for many module registers - the value after reset is always zero, for example. In the following description, module register properties are mentioned only if a property deviates from the following default properties.

| Module register properties | Default property for most module registers |
|---|---|
| Access | Read/write |
| Value after reset | 0 or undefined (e.g. release number) |
| Takes effect | Immediately |
| Write access | Always |
| Data type | Integer |

## Numerical formats

The numerical formats used in this document are listed in the table below:

| Notation | Numerical format |
|---|---|
| 100 | Decimal |
| 0x100 | Hexadecimal |
| 0b100 | Binary |

## JetSym sample programs

The notation for sample programs used in this document is listed in the table below:

| Notation | Description |
|---|---|
| `Var, When, Task` | Keyword |
| `BitClear();` | Commands |
| `100 0x100 0b100` | Constant numerical values |
| `// This is a comment` | Comment |
| `// ...` | Further program processing |

# 10.1 Memories - Overview

**Introduction**

The JC-360 features several types of program and data memories. There is volatile and non-volatile memory. Volatile memory loses its content at switching off. Non-volatile memory keeps its content even when the power supply is off.

The memory is located directly in the CPU or in separate memory or I/O modules.

This chapter gives an overview of the available memory.

**Contents**


| Topic | Page |
|---|---|
| Operating system memory | 218 |
| File system memory | 219 |
| Application program memory | 220 |
| Memory for volatile application program variables | 221 |
| Memory for non-volatile application program registers | 222 |
| Memory for non-volatile application program variables | 223 |
| Registers on I/O modules | 224 |
| Memory for non-volatile registers on the backplane module | 225 |
| Special registers | 226 |
| Inputs and outputs | 227 |
| Flags | 228 |

## Operating system memory

| | |
|---|---|
| **Introduction** | The OS is stored to a non-volatile flash memory in the CPU. Therefore, the OS can be executed immediately after the device is powered up. |
| **Properties** | ▪ Internal flash memory for storing OS data<br>▪ Internal volatile RAM for storing OS data |
| **Memory access** | ▪ The user is not allowed to directly access the OS memory.<br>▪ To modify the OS, it must be updated. |
| **Related topics** | ▪ **OS update** (see page 606) |

# File system memory

| | |
|---|---|
| **Introduction** | The file system memory is for storing data and program files. |
| **Properties** | ▪ Internal flash disk and SD memory card<br>▪ Non-volatile<br>▪ Slow access: Milliseconds up to seconds<br>▪ A limited number of write/erase cycles: approx. 1 million<br>▪ Internal flash disk size: 4 MB<br>▪ SD card size: 32 MB ... 4 GB |
| **Memory access** | ▪ By operating system<br>▪ By JetSym<br>▪ Via FTP connection<br>▪ By the e-mail client<br>▪ By browser (via HTTP server)<br>▪ By means of file commands from within the application program |

# Application program memory

**Introduction**

By default, the application program is uploaded from JetSym to the controller and is stored to it.

**Properties**

- Stored as file within the file system
- Default directory */app*
- Files may also be stored to other directories (or on SD card)

**Memory access**

- By operating system
- By JetSym
- Via FTP connection
- By means of file commands from within the application program

**Related topics**

- **Application program** (see page 617)

# Memory for volatile application program variables

| | |
|---|---|
| **Introduction** | Volatile variables are used to store data which need not be maintained when the JC-360 is de-energized. |
| **Properties** | ▪ Global variables which are not assigned to permanent addresses (not %VL or %RL)<br>▪ Local variables<br>▪ Variables are stored in a compact way.<br>▪ Variables are initialized with value 0 when they are created. |
| **Memory access** | ▪ By JetSym<br>▪ From the application program |

## Memory for non-volatile application program registers

| | |
|---|---|
| **Introduction** | Non-volatile registers let you store data which must be maintained when the JC-360 is de-energized. |
| **Properties** | ▪ Global variables which are assigned to permanent addresses (%VL)<br>▪ Register variables always occupy 4 bytes.<br>▪ Register variables are not initialized by the operating system.<br>▪ Number of register variables: 60,000 (120,000 with option -R)<br>▪ Register numbers: 1,000,000 ... 1,059,999 (up to 1,119,999 with option -R) |
| **Memory access** | ▪ By JetSym<br>▪ By the e-mail client<br>▪ By browser (via HTTP server)<br>▪ From displays and HMIs<br>▪ From the application program<br>▪ From other controllers |

# Memory for non-volatile application program variables

| | |
|---|---|
| **Introduction** | Non-volatile variables let you store data which must be maintained when the JC-360 is de-energized. |
| **Properties** | ▪ Global variables which are assigned to permanent registers (%RL)<br>▪ Variables are stored in a compact way.<br>▪ Size: 240,000 bytes (480,000 with option -R)<br>▪ Register numbers: 1,000,000 ... 1,059,999 (up to 1,119,999 with option -R) |
| **Memory access** | ▪ By JetSym<br>▪ From displays and HMIs<br>▪ From the application program |

## Registers on I/O modules

| | |
|---|---|
| **Introduction** | These registers are located on modules connected to the JX2 or JX3 system bus. |
| **Properties** | ▪ Global variables which are assigned to permanent addresses (%VL)<br>▪ Type depending on the module<br>▪ Register numbers on JX3 system bus: 100020000 ... 100179999<br>▪ Register numbers on JX2 system bus: 200002000 ... 100019999 |
| **Memory access** | ▪ By JetSym<br>▪ By the e-mail client<br>▪ By browser (via HTTP server)<br>▪ From HMIs<br>▪ From the application program<br>▪ From other controllers |

# Memory for non-volatile registers on the backplane module

| | |
|---|---|
| **Introduction** | These registers are located on the backplane module of the controller. |
| **Properties** | ▪ Global variables which are assigned to permanent addresses (%VL)<br>▪ Integer registers<br>▪ Slow access: Milliseconds<br>▪ Limited number of write/erase cycles: Approx. 10,000<br>▪ Number of registers: 128<br>▪ Register numbers: 108100 ... 108227 |
| **Memory access** | ▪ By JetSym<br>▪ By the e-mail client<br>▪ By browser (via HTTP server)<br>▪ From HMIs<br>▪ From the application program<br>▪ From other controllers |

## Special registers

| | |
|---|---|
| **Introduction** | Special registers let you control OS functions and retrieve status information. |
| **Properties** | ▪ Global variables which are assigned to permanent addresses (%VL)<br>▪ When the operating system is launched, special registers are initialized using default values.<br>▪ Register numbers: 100000 ... 999999 |
| **Memory access** | ▪ By JetSym<br>▪ By the e-mail client<br>▪ By browser (via HTTP server)<br>▪ From HMIs<br>▪ From the application program<br>▪ From other controllers |

# Inputs and outputs

| | |
|---|---|
| **Introduction** | Inputs and outputs are 1-bit variables. This means they can either have the value TRUE or FALSE. |
| **Properties of virtual inputs/outputs** | ▪ Global variables assigned to permanent addresses (%IX, %QX)<br>▪ Used for RemoteScan via Modbus/TCP<br>▪ Amount: 16,000<br>▪ I/O numbers: 20001 ... 36000 |
| **Properties of digital inputs/outputs** | ▪ Global variables assigned to permanent addresses (%IX, %QX)<br>▪ Located on modules connected to the JX2 or JX3 system bus<br>▪ I/O numbers on the JX3 system bus: 100000201 ... 100001716<br>▪ I/O numbers on the JX2 system bus: 200000201 ... 200002416<br>▪ I/O numbers of remote devices connected to a JX3-BN-ETH: 1nnn010201 ... 1nnn011716 (nnn = GNN) |
| **Memory access** | ▪ By JetSym<br>▪ By the e-mail client<br>▪ By browser (via HTTP server)<br>▪ From displays and HMIs<br>▪ From the application program |

# Flags

| | |
|---|---|
| **Introduction** | Flags are one-bit operands. This means they can either have the value TRUE or FALSE. |
| **Properties of user flags** | ▪ Global variables which are assigned to permanent addresses (%MX)<br>▪ Non-volatile<br>▪ Amount: 256<br>▪ Flag numbers: 0 ... 255 |
| **Properties of overlaid user flags** | ▪ Global variables which are assigned to permanent addresses (%MX)<br>▪ Non-volatile<br>▪ Overlaid by registers 1000000 ... 1000055<br>▪ Amount: 1,792<br>▪ Flag numbers: 256 ... 2047 |
| **Properties of special flags** | ▪ Global variables which are assigned to permanent addresses (%MX)<br>▪ When the operating system is launched, special flags are initialized using their default values.<br>▪ Amount: 256<br>▪ Flag numbers: 2048 ... 2303 |
| **Memory access** | ▪ By JetSym<br>▪ By the e-mail client<br>▪ By browser (via HTTP server)<br>▪ From HMIs<br>▪ From the application program |

# 10.2 Register and I/O numbers with a JC-3xx

**Introduction**

Controllers and modules produced by Jetter AG offer a host of functions which can be accessed by the user via registers. A unique number is assigned to each register and each digital input or output.

**Applying a register number**

Register numbers are applied in the following cases:

- You want to read or write to a module register in the JetSym setup.
- You want to declare a module register a variable in the JetSym application program.
- You want to declare a module register a tag in JetViewSoft

**Applying an I/O number**

I/O numbers are applied in the following cases:

- You want to read from a digital input in the JetSym setup.
- You want to read or write to a digital output in the JetSym setup.
- You want to declare a digital input or output a variable in the JetSym application program.
- You want to declare a digital input or output a tag in JetViewSoft.

**Contents**


| Topic | Page |
|---|---|
| Registers and module registers | 230 |
| Register and I/O numbers of JX3 modules connected to a JC-3xx | 232 |
| Register numbers of JX2 slave modules connected to the JX2 system bus | 233 |
| Registers and I/O numbers of JX2-I/O modules on the JX2 system bus | 234 |
| Register and I/O numbers of IP67-I/O modules on the JX2 system bus | 235 |
| Registers and I/O numbers of CANopen® modules on the JX2 system bus | 236 |
| Register and I/O numbers of JX3 modules connected to a JX3-BN-ETH | 237 |
| Registers and I/O numbers of JX3 nodules from the JX3-BN-ETH perspective | 239 |

# Registers and module registers

**Module registers - Definition**

Module registers are the data interface of a JX3 module. Module registers let you read process, configuration and diagnostics data from the JX3 module, or write such data to it.

- The module register number within a module is unique.
- This unique register number lets you access a specific module register within the system.

**Registers - Definition**

There are several ways to access registers directly:

- Via an application program
- Via a JetSym setup pane
- Via a visualization application

The register number within the system is unique.

**Definition - Global Node Number**

The Global Node Number (GNN) is an ID number to identify Jetter controllers (e.g. bus nodes, bus nodes) within an Ethernet network.

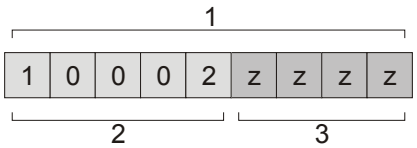
- The GNN within a network has to be unambiguous for each Jetter device.
- The JetSym Hardware Manager automatically assigns the GNN during configuration.
- The value range of the GNN within a project is 000 ... 199.
- The controller has always got GNN 000.

**Example - Module registers**

Via module register 9 the OS revision of a JX3-AI4 module can be accessed.

**Registers - Example**

A JX3-AI4 module is connected to a controller JC-3xx. The module number of this module is 2.



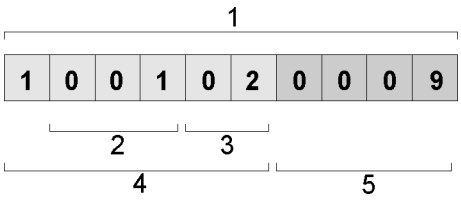

| Number | Element | Description |
|---|---|---|
| **1** | Register number | Supports direct access |
| **2** | Register prefix | 10002: Assigned to the first JX3 module connected to a JC-3xx controller |
| **3** | Module register number | zzzz = 0009: OS version of JX3-AI4 |

In the setup pane of JetSym you can directly read out the operating system version 1.2.0.0 from register number 100020009.

| | Name | Number | Content | Type |
|---|---|---|---|---|
| 40 | 100020009 | 100020009 | 1.2.0.0 | |
| 41 | | | | |
| 42 | | | | |

**Example: Registers on the Ethernet system bus**

A JX3-AI4 module is connected to a bus node JX3-BN-ETH. The module number of the JX3 module is 2. The bus node has got the ID (GNN) 001.




| Number | Element | Description |
|---|---|---|
| **1** | Register number | Supports direct access |
| **2** | Bus node ID, GNN | 001: ID of the first JX3-BN-ETH |
| **3** | Module number | 02: The first JX3-I/O module connected to the JX3-BN-ETH |
| **4** | Register prefix | 100102 |
| **5** | Module register number | 0009: OS version of JX3-AI4 |

In the setup pane of JetSym you can directly read out the operating system version 1.4.0.0 from register number 1001020009.

| | Name | Number | Content | Type |
|---|---|---|---|---|
| 1 | 1001020009 | 100102000 | 1.4.0.0 | |
| 2 | | | | |

# Register and I/O numbers of JX3 modules connected to a JC-3xx

**Module numbers in a JX3 station**

To determine I/O module numbers within a JX3 station, proceed as follows:

- Count the module numbers left-to-right, starting with 1.
- Leave out the power supply module JX3-PS1.

**Register numbers for JX3 modules**

Register numbers for JX3 modules connected to a JC-3xx consist of the following elements:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | x | x | z | z | z | z |

| Element | Description | Value range |
|---|---|---|
| **xx** | Number of the module within the JX3 station | 02 ... 17 |
| **zzzz** | Module register number | 0000 ... 9999 |

**I/O numbers for JX3 modules**

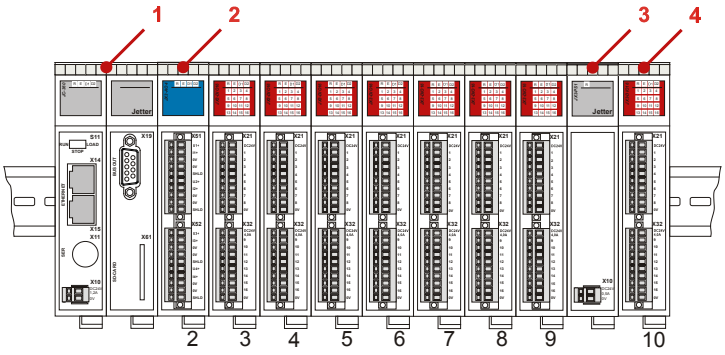
I/O numbers for JX3 modules connected to a JC-3xx consist of the following elements:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | x | x | z | z |

| Element | Description | Value range |
|---|---|---|
| **xx** | Number of the module within the JX3 station | 02 ... 17 |
| **zz** | Module-specific I/O number | 1 ... 16 |

**Example**

Several JX3 modules have been connected to a JC-3xx controller.




| Number | Module | Module number | Register | I/O |
|---|---|---|---|---|
| **1** | JC-3xx | 1 | Refer to documentation on JC-3xx | |
| **2** | JX3-AO4 | 2 | 10002zzzz | 1000002zz |
| **3** | JX3-PS1 | - | - | - |
| **4** | JX3-DIO16 | 10 | 10010zzzz | 1000010zz |

# Register numbers of JX2 slave modules connected to the JX2 system bus

**Slave module numbers of JX2 slave modules**

To determine the slave module numbers of intelligent JX2 slave modules and JetMoves on the JX2 system bus of the JC-3xx, proceed as follows:

- Count the JX2 slave modules left-to-right, starting with 2.
- Leave out the power supply module JX2-PS1.
- Leave out non-intelligent JX2-I/O modules.

**Register numbers for JX2 slave modules**

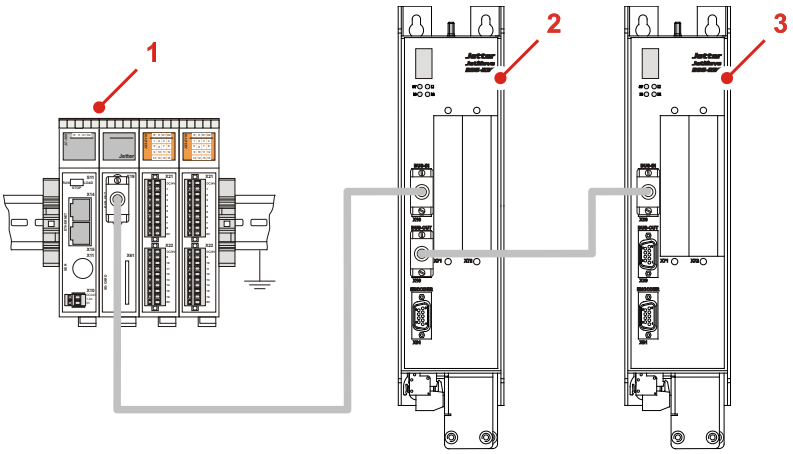
Register numbers for JX2 slave modules connected to the JX2 system bus of the JC-3xx consist of the following elements:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 2 | 0 | 0 | 0 | x | x | z | z | z |

| Element | Description | Value range |
|---|---|---|
| **xx** | Slave module number + 10 | 12 ... 27 |
| **zzz** | Module register number | 000 ... 999 |

**Example**

Several JM-200 drives are connected to a JC-3xx controller.




| Number | Module | Slave module number | Register |
|---|---|---|---|
| **1** | JC-3xx | 1 | Refer to documentation on JC-3xx |
| **2** | JM-206 | 2 | 200012zzz |
| **3** | JM-206 | 3 | 200013zzz |

# Registers and I/O numbers of JX2-I/O modules on the JX2 system bus

**I/O module numbers of JX2-I/O modules**

To determine the I/O module numbers of JX2-I/O modules on the JX2 system bus of the JC-3xx, proceed as follows:

- Count the JX2-I/O modules left-to-right, starting with 2.
- Leave out the intelligent JX2 slave modules and JetMoves.
- Leave out the power supply module JX2-PS1.

**Register numbers for JX2-I/O modules**

Register numbers for JX2-I/O modules connected to the JX2 system bus of a JC-3xx consist of the following elements:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 2 | 0 | 0 | 0 | 0 | 3 | x | x | z |

| Element | Description | Value range |
|---|---|---|
| **xx** | I/O module number minus 2 | 00 ... 22 |
| **z** | Module register number | 0 ... 9 |

**I/O numbers for JX2-I/O modules**

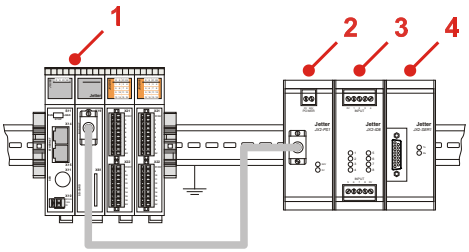
I/O numbers for JX2-I/O modules connected to the JX2 system bus of a JC-3xx consist of the following elements:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 2 | 0 | 0 | 0 | 0 | x | x | z | z |

| Element | Description | Value range |
|---|---|---|
| **xx** | Module-specific I/O module number | 02 ... 24 |
| **zz** | Module-specific I/O number | 1 ... 16 |

**Example**

Several JX2-I/O modules are connected to a JC-3xx controller.




| Number | Module | I/O module number | Register | I/O |
|---|---|---|---|---|
| **1** | JC-3xx | 1 | Refer to documentation on JC-3xx. | |
| **2** | JX2-PS1 | - | - | - |
| **3** | JX2-ID8 | 2 | 20000300z | 2000002zz |
| **4** | JX2-CNT | 3 | 20000301z | 2000003zz |

# Register and I/O numbers of IP67-I/O modules on the JX2 system bus

**I/O module numbers of IP67-I/O modules**

To determine the I/O module numbers of IP67-I/O modules on the JX2 system bus of the JC-3xx, proceed as follows:

- Set the I/O module numbers by means of the addressing switch located on the module itself.
- LioN-S and LJX7-CSL modules are counted among IP67-I/O modules.

**Register numbers for IP67-I/O modules**

Register numbers for IP67-I/O modules connected to the JX2 system bus of a JC-3xx consist of the following elements:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 2 | 0 | 0 | 0 | 0 | 3 | x | x | z |

| Element | Description | Value range |
|---|---|---|
| **xx** | I/O module number minus 2 | 00 ... 22 |
| **z** | Module register number | 0 ... 9 |

**I/O numbers for IP67-I/O modules**

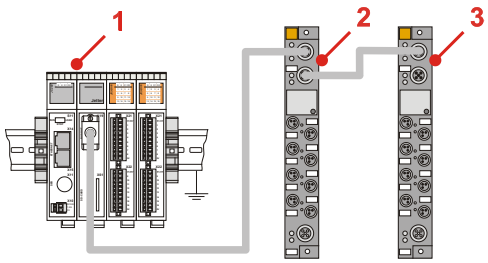
I/O numbers for IP67-I/O modules connected to the JX2 system bus of a JC-3xx consist of the following elements:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 2 | 0 | 0 | 0 | 0 | x | x | z | z |

| Element | Description | Value range |
|---|---|---|
| **xx** | Module-specific I/O module number | 02 ... 24 |
| **zz** | Module-specific I/O number | 1 ... 16 |

**Example**

Several IP67-I/O modules are connected to a JC-3xx controller.




| Number | Module | I/O module number | Register | I/O |
|---|---|---|---|---|
| **1** | JC-3xx | 1 | Refer to documentation on JC-3xx. | |
| **2** | LioN-S | 2 | 20000300z | 2000002zz |
| **3** | LioN-S | 3 | 20000301z | 2000003zz |

# Registers and I/O numbers of CANopen® modules on the JX2 system bus

**I/O module numbers of CANopen® modules**

To determine the I/O module numbers of CANopen® modules on the JX2 system bus of the JC-3xx, proceed as follows:

- In most cases, the I/O module numbers correspond to the node ID of the CANopen® module.
- Exceptions: SMC EX120 and Lenze frequency inverters

**Register numbers for CANopen® modules**

Register numbers for CANopen® modules connected to the JX2 system bus of the JC-3xx consist of the following elements:

| 2 | 0 | 0 | 0 | 0 | x | x | z | z |
|---|---|---|---|---|---|---|---|---|

| Element | Description | Value range |
|---|---|---|
| **xx** | I/O module number | 70 ... 79 |
| **zz** | Module register number | 00 ... 99 |

**I/O numbers for CANopen® modules**

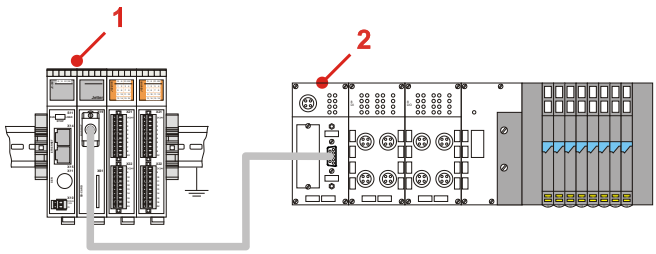
I/O numbers for CANopen® modules connected to the JX2 system bus of a JC-3xx consist of the following elements:

| 2 | 0 | 0 | 0 | 0 | x | x | z | z |
|---|---|---|---|---|---|---|---|---|

| Element | Description | Value range |
|---|---|---|
| **xx** | Module-specific I/O module number | 70 ... 79 |
| **zz** | Module-specific I/O number | 1 ... 64 |

**Example**

A CANopen® module is connected to a JC-3xx controller.




| Number | Module | I/O module number | Register | I/O |
|---|---|---|---|---|
| **1** | JC-3xx | 1 | Refer to documentation on JC-3xx | |
| **2** | Festo CPX | 2 | 2000070zz | 2000070zz |

# Register and I/O numbers of JX3 modules connected to a JX3-BN-ETH

**Global Node Numbers on the Jetter Ethernet system bus of a JX3-BN-ETH**

JetSym Hardware Manager assigns a Global Node Number to the bus node JX3-BN-ETH on the Jetter Ethernet system bus.

**Register numbers for JX3 modules**

The register number for JX3 modules at the Ethernet bus node consists of the following elements:

| 1 | n | n | n | x | x | z | z | z | z |
|---|---|---|---|---|---|---|---|---|---|

| Element | Description | Value range |
|---|---|---|
| **nnn** | Global Node Number of the JX3-BN-ETH on the Ethernet system bus | 001 ... 199 |
| **xx** | Number of the module within the JX3 station | 02 ... 17 |
| **zzzz** | Module register number | 0000 ... 9999 |

**I/O numbers for JX3 modules**

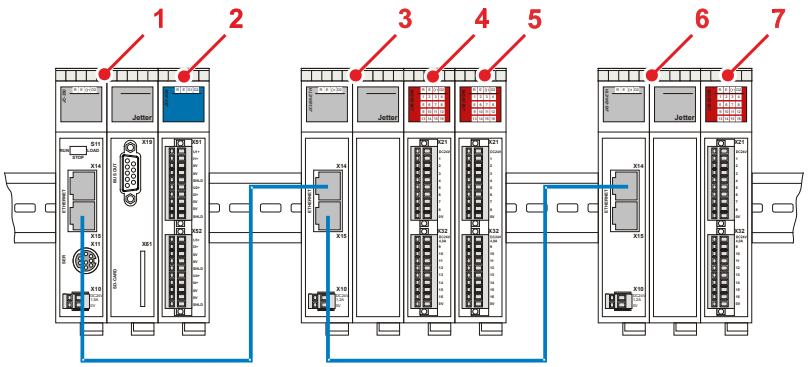
The I/O number for JX3 modules connected to an Ethernet bus node consists of the following elements:

| 1 | n | n | n | 0 | 1 | x | x | z | z |
|---|---|---|---|---|---|---|---|---|---|

| Element | Description | Value range |
|---|---|---|
| **nnn** | Global Node Number of the JX3-BN-ETH on the Ethernet system bus | 001 ... 199 |
| **xx** | Number of the module within the JX3 station | 02 ... 17 |
| **zz** | Module-specific I/O number | 1 ... 16 |

**Example**

Two bus nodes JX3-BN-ETH are connected to a JC-3xx controller.




| Number | Module | Module number | GNN | Register | I/O |
|---|---|---|---|---|---|
| **1** | JC-3xx | 1 | - | Refer to JC-3xx manual | |
| **2** | JX3-AO4 | 2 | - | 10002zzzz | 1000002zz |
| **3** | JX3-BN-ETH | - | 1 | Refer to JX3-BN-ETH manual | |
| **4** | JX3-DIO16 | 2 | - | 100102zzzz | 10010102zz |
| **5** | JX3-DIO16 | 3 | - | 100103zzzz | 10010103zz |
| **6** | JX3-BN-ETH | - | 2 | Refer to JX3-BN-ETH manual | |
| **7** | JX3-DIO16 | 2 | - | 100202zzzz | 10020102zz |

# Registers and I/O numbers of JX3 nodules from the JX3-BN-ETH perspective

**Application example**

Explicit data transfer using `NetCopy()`.

**Module numbers in a JX3 station**

To determine module numbers in a JX3 station, proceed as follows:

- Count the JX3-I/O modules left-to-right, starting with 1.
- Leave out the power supply module JX3-PS1.

**Register numbers for JX3 modules**

From the perspective of the Ethernet bus node, the register number consists of the following elements:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | x | x | z | z | z | z |

| Element | Description | Value range |
|---|---|---|
| **xx** | Number of the module within the JX3 station | 02 ... 17 |
| **zzzz** | Module register number | 0000 ... 9999 |

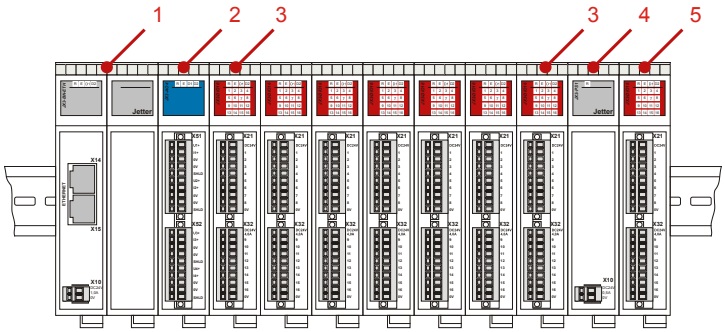
**I/O numbers for JX3-modules**

From the perspective of the Ethernet bus node, the I/O number consists of the following elements:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | x | x | z | z |

| Element | Description | Value range |
|---|---|---|
| **xx** | Number of the module within the JX3 station | 02 ... 17 |
| **zz** | Module-specific I/O number | 1 ... 16 |

**Example of a JX3 station at a JX3-BN-ETH**




| Number | Module | Module number | Register | I/O |
|---|---|---|---|---|
| **1** | JX3-BN-ETH | 1 | Refer to JX3-BN-ETH manual | |
| **2** | JX3-AO4 | 2 | 10002zzzz | 1000002zz |
| **3** | JX3-DIO16 | 3 ff. | 10003zzzz | 1000003zz |
| **4** | JX3-PS1 | - | - | - |
| **5** | JX3-DIO16 | 10 | 10010zzzz | 1000010zz |

# 10.3 Jetter Ethernet system bus

**Introduction**

The Jetter Ethernet system bus has been designed for data exchange between the following devices via standard Ethernet.

- Programming device
- Controllers
- Bus node
- Communication modules

**Data interchange**

The Jetter Ethernet system bus makes a difference between the cyclic and acyclic data interchange between communication participants. Both kinds of data interchange can be executed simultaneously within a network.

| Data exchange | Properties |
|---|---|
| Cyclic | ■ **Architecture:** Publish/subscribe<br>■ **Nodes:** Controllers, bus nodes and communication modules<br>■ **Access:** Automatically by OS<br>■ **Access time:** Fast, deterministic<br>■ **Data:** Registers, inputs/outputs<br>■ **Configuration:** Hardware Manager in JetSym<br>■ **Reach:** Subnet |
| Acyclic | ■ **Architecture:** Client/server<br>■ **Client:** PC and controllers<br>■ **Server:** PC, controllers, bus nodes and communication modules<br>■ **Data:** E.g. registers, inputs/outputs, STX variables, application program<br>■ **Access:** PC or application program<br>■ **Access time:** Depending on the reaction time of the server<br>■ **Configuration:** Only when using network registers<br>■ **Reach:** International |

**Minimum requirements**

The device is operated in a system consisting of various components by Jetter AG. In order to ensure proper interaction of these components, the operating system used and the programming tool JetSym must have at least the release numbers listed below.

| Component | As of version |
|---|---|
| JC-310-JM | V. 1.22.0.00 |
| JC-340 | V. 1.22.0.00 |
| JC-350 | V. 1.22.0.00 |
| JC-360 | V. 1.22.0.00 |
| JC-360MC | V. 1.22.0.00 |
| JC-365 | V. 1.26.0.00 |
| JC-365MC | V. 1.26.0.00 |
| JC-940MC | V. 1.06.0.20 |
| JC-945MC | V. 1.01.0.00 |
| JX3-BN-ETH | V. 1.18.0.02 |
| JX3-COM-EIPA | V. 1.01.0.00 |
| JX3-COM-PND | V. 1.03.0.06 |
| JM-200-ETH | V. 1.22.0.00 |
| JetSym | V. 5.1.2 |

**Contents**


| Topic | Page |
|---|---|
| The Global Node Number | 242 |
| Acyclic data interchange | 243 |
| Cyclic data interchange | 259 |
| Hardware Manager | 272 |
| Error handling at the Jetter Ethernet system bus | 274 |
| JetIPScan - Register description | 279 |
| Administrating the connections of the JetIP/TCP and STX debug server | 290 |
| Executing an ARP request | 294 |
| JetSync blockage | 296 |

# The Global Node Number

**Definition - Global Node Number**

The Global Node Number (GNN) is an ID number to identify Jetter controllers (e.g. controllers, bus nodes) within an Ethernet network.

- The GNN within a network has to be unambiguous for each Jetter device.
- The JetSym Hardware Manager automatically assigns the GNN during configuration.
- The value range of the GNN within a project is 000 ... 199.
- The controller has always got GNN 000.

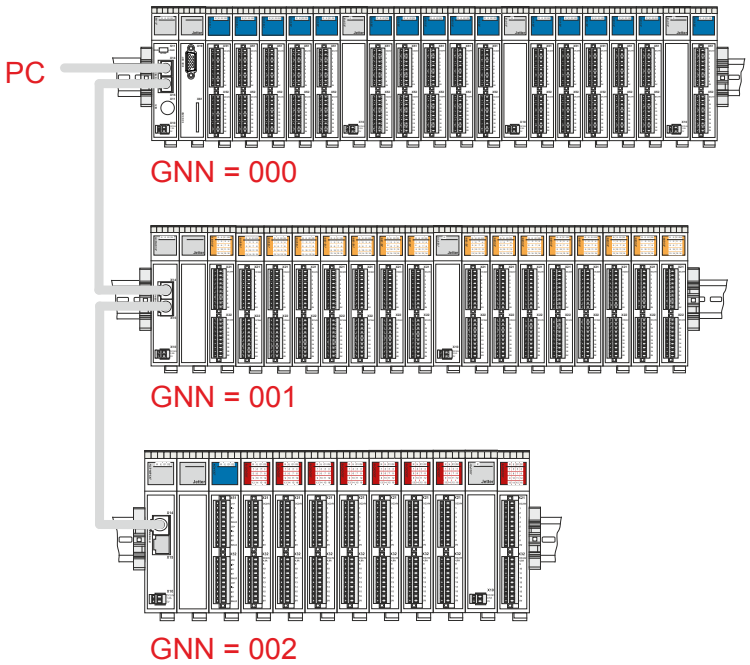
**Using the GNN**

The Global Node Number is used in the following applications:

- Register number for network registers
- Identification of publications and subscriptions at cyclic data interchange
- Identification of nodes at automatic network configuration (NetConsistency)

**Networking example**

The following illustration shows networking of a possible JX3 system with a JC-3xx and two JX3-BN-ETH.



## 10.3.1 Acyclic data interchange

**Introduction**

This chapter covers acyclic data interchange on the Jetter Ethernet system bus.

**Properties**

Acyclic data interchange on the Jetter Ethernet system bus can be characterized as follows:

| Property | Description |
|---|---|
| Architecture | Client/server<br>■ Data interchange is initiated by the client.<br>■ The server gives a response to the request made by the client.<br>■ Usage of unicast frames<br>■ Network access is made once. |
| Client | ■ JetSym:<br>Programming and debugging of application programs<br>■ JetViewSoft:<br>Setting up a visualization application<br>■ Controllers:<br>Data interchange out of the application program (NetCopy..., NetBit..., network register) |
| Server | ■ PC:<br>E.g. for database applications<br>■ Controllers, bus nodes and communication modules:<br>E.g. variable or debugging server |
| Data | ■ PC:<br>Registers, inputs/outputs, STX variables, application program<br>■ Controllers, bus nodes and communication modules:<br>Registers, STX variables |
| Access time | ■ It depends on the data transfer time and on the server's processing time |
| Configuration | ■ Network registers:<br>Easy configuration in the application program<br>■ Else, both client and server are completely configured by the operating system. |
| Reach | ■ Using TCP/IP and UDP/IP frames allow for data interchange exceeding the limits of one's own subnet. |

**Client**

Below, programming the client in the controllers is described. In doing so, the following topics are dealt with:

- Transferring variable/register sets (command group NetCopy())
- Setting and clearing register bits (command group NetBit())
- Transmitting individual register values (network registers)

**Examples of the application**

- Event-triggered data interchange
- Parameterization
- Configuration

**Used protocol**

The client of the controller uses the JetIP protocol based on UDP/IP for data transfer.

**Server**

The server functions do not require any programming or configuration by the user.

**Protocols**

Acyclic data interchange on the Jetter system bus can be established by the following protocols:

- XCOM protocol by Jetter AG
- JetIP protocol by Jetter AG
- UDP/IP
- TCP/IP
- IPv4

**Contents**


| Topic | Page |
|---|---|
| Command group NetCopy() | 245 |
| Command group NetBit() | 247 |
| Network registers | 248 |
| Registers located on JX3 modules | 250 |
| Indirect addressing of remote modules | 252 |
| Addressing with variable destination window | 254 |
| Register description - Acyclic data interchange | 256 |

# Command group NetCopy()

**Introduction**

The NetCopy command is a versatile tool for data interchange between Jetter products via Ethernet.

The NetCopy command lets you copy the following data:

- Register values
- Values of register blocks
- Variable values
- Values of variable blocks

**Advantages of NetCopy**

Advantages of NetCopy commands as compared with the use of network registers:

- Within the command, you can directly specify any valid IP address.
- Within the command, you can directly specify any valid IP port.
- The entire register address range of a remote node can be directly addressed.
- By means of one command, a large register set or, in case NetCopyList is applied, a large number of registers can be copied.
- The result of the copying process can be evaluated directly.

**Access via NetCopy**

NetCopy functions with the following nodes:

- Controllers
- Bus node
- Communication modules
- PC

To access other nodes, use the NetCopy command as follows:

| If ... | ... then ... |
|---|---|
| ... you wish to copy data from the controller to another node, | ... use the following commands:<br>■ NetCopyRegToReg<br>■ NetCopyVarToReg<br>■ NetCopyList |
| ... you wish to copy data from another node to the controller, | ... use the following commands:<br>■ NetCopyRegFromReg<br>■ NetCopyVarFromReg<br>■ NetCopyList |

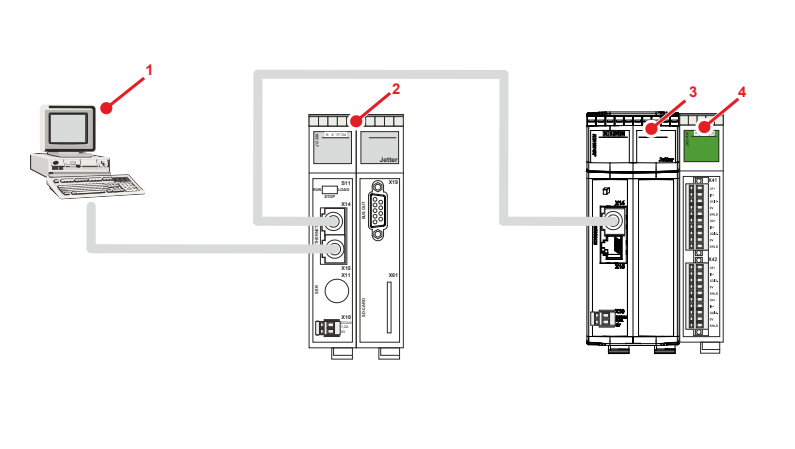
**Parameters of the NetCopy commands**

For detailed information on the parameters, refer to the JetSym help.

**NetCopy - Example featuring a bus node**

As you can see in the following illustration, a controller JC-3xx is connected to a PC. The bus node JX3-BN-ETH of IP address 192.168.10.2 is connected to a peripheral module JX3-AI4.

This example describes how to access the module registers of the peripheral module JX3-AI4 in acyclic mode.




| Number | Part | Description |
|---|---|---|
| **1** | PC | PC with JetSym |
| **2** | JC-3xx | Controller |
| **3** | JX3-BN-ETH | Bus node |
| **4** | JX3-AI4 | Peripheral module with analog inputs |

### Task

When an event occurs, user scaling of analog input 1 is to be changed.

### Solution

The NetCopy command causes values from application program variables to which the user scaling parameters have been stored to be copied to the corresponding registers of the JX3-AI4.

The register number of the peripheral module is seen from the perspective of the JX3-BN-ETH:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | x | x | z | z | z | z |

with

- xx = 02: First module on the JX3-BN-ETH
- zzzz = 1124 through 1127: Parameter registers of the JX3-AI4 user scaling

```
// Copy values from the local array to the JX3-AI4
nResult := NetCopyVarToReg(IP#192.168.10.2, anParam,
                           100021124, 16, 3, 1);
```

# Command group NetBit()

**Introduction**

The NetBit command is an all-purpose tool to set or clear register bits of Jetter products. The Jetter products are interconnected via an Ethernet network.

**Advantages of NetBit**

NetBit commands let you both set and clear bits in one go.

Simulating NetBit commands by means of NetCopy commands:

- A NetCopy command lets you copy the register value from the remote node to the local controller.
- A NetCopy command lets you change the state of the bits on the local controller as desired.
- Another NetCopy command lets you copy the register value to the remote node again.

For this, several commands are required. Thus, a register value may be changed during this action by an application program running on the remote controller. The second NetCopy command will then overwrite this value again. There is an undefined data condition, which is prevented by the NetBit functions.

Further advantages of NetBit commands as compared with the use of network registers:

- Within the command, you can directly specify any valid IP address.
- Within the command, you can directly specify any valid IP port.
- The entire register address range of a remote node can be directly addressed.
- The result of executing this command can be evaluated directly.

**Access via NetBit**

NetBit functions with the following nodes:

- Controllers
- Bus node
- Communication modules

To access other nodes, use the command NetBit as follows:

| If ... | ... then ... |
|---|---|
| you wish to set register bits for another node, | use the command<br>■ NetBitSetReg |
| you wish to clear register bits of another node, | use the command<br>■ NetBitClearReg |

**Parameters of the NetBit commands**

For detailed information on the parameters, refer to the JetSym help.

# Network registers

**Introduction**

The network registers let you access in transparent mode registers of remote nodes.

**Advantages**

Advantages of network registers as compared with NetCopy commands:

- Network registers are used just like any other registers in the application program.
- If programs or parts of programs are used for local and distributed applications, a program description is not needed.

**Restrictions**

The following restrictions apply to network registers as compared with NetCopy commands:

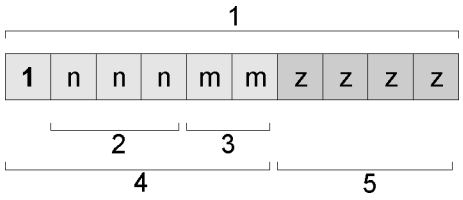
- IP address and IP port of the remote node must be set separately.
- Only part of the register address range of the remote nodes can be accessed directly.
- The outcome of the network access (diagnostics) cannot be logged directly.

**Properties**

If you access network registers of cyclic data interchange, the controller does not carry out acyclic network register access. The controller accesses the locally stored cyclic data.

**Addressing scheme**

The addressing scheme for network registers is as follows:




| No. | Element | Description |
|---|---|---|
| **1** | Register number | Supports direct access |
| **2** | First part of register prefix: Bus node ID, GNN | nnn = 001 ... 199: ID of the network node, referred to as Global Node Number |
| **3** | Second part of register prefix: Number of the function module | mm = 02 ... 17: Number of the JX3 module of a remote node<br>mm = 98: Indirect addressing of the register of a remote node<br>mm = 99: Addressing the variable destination window of a remote node |
| **4** | Part 1 + 2: Register prefix | 1nnnmm: The prefix is preceded by a leading ONE. |
| **5** | Module register number | zzzz = 0000 ... 9999 |

**IP address and IP port**

Before using a network register, the IP addresses and IP ports of the remote network nodes must be written to two tables in the local register array.

| If ... | ... then ... |
|---|---|
| ... you carry out network configuration in the JetSym Hardware Manager, | ... these tables are generated automatically, see file **ModConfig.da** below. |
| ... you do not carry out network configuration in the Hardware Manager, | ... you must generate the tables in your application program. |

Content indexing of the tables is done via GNN of the node in the first part of the register prefix (2).

| Register | Value range | Properties |
|---|---|---|
| 235000 + GNN | 235000 ... 235199 | Register table for **IP addresses** |
| 235400 + GNN | 235400 ... 235599 | Register table for **IP ports** |

Note on the contents of the table:

- GNN = Global Node Number in the range 000 ... 199

**File *ModConfig.da***

When you download the configuration files, the Hardware Manager transfers the file **ModConfig.da** to the controller.

The OS of the controller loads this file when the controller is energized or when the corresponding command is automatically issued by the Hardware Manager after download.

The file **ModConfig.da** lists registers with their corresponding values. The OS enters the corresponding values into these registers.

This file also holds the IP addresses (register 235000 + GNN) and port numbers (register 235400 + GNN) of the nodes on the network.

It is no longer required to enter values into registers via application program.

# Registers located on JX3 modules

**Introduction**

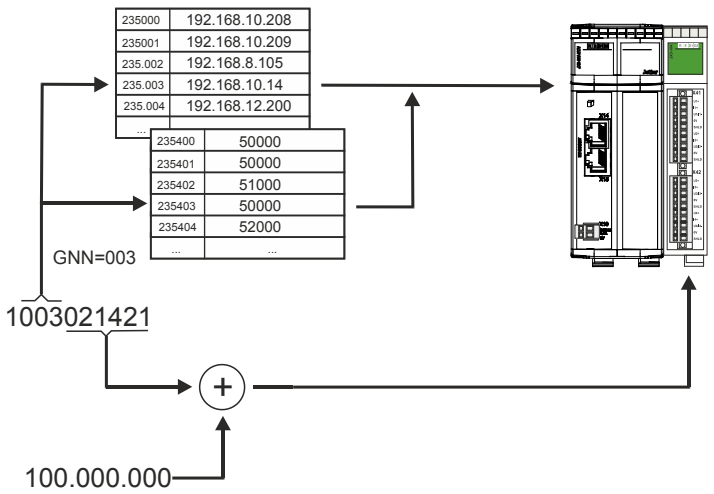
The controller handles access via network registers to module registers of JX3 modules of a remote node (second part of the register prefix mm = 02 ... 17) in a specific way:

| If ... | ... then ... |
|---|---|
| ... the network register has been configured for cyclic data interchange, | ... the controller accesses the locally stored register value. |
| ... the network register has not been configured for cyclic data interchange, | ... the controller executes acyclic network access. |

**Acyclic network access**

For acyclic register access to a remote JX3 module, the controller uses the first part of the register prefix as an index to the tables containing the IP addresses and IP ports. The values read out of these tables are used by the controller to address the bus node in the network. For further information on registers 235000 ff, please refer to **Network registers - IP address and IP port** (see page 248).

The controller adds the register offset for the JX3 system bus of a remote node (100,000,000) to the second part of the register prefix and the module register number (no. 3 and 5 in the **addressing scheme** (see page 248)). The controller uses the resulting number to address the register.



**Action**

If you want to access the JX3 module register of a remote network node using register addresses as of 1 billion, proceed as follows:

| Step | Action |
|---|---|
| **1** | Enter the **IP address** of the remote network node into register **235.000 + GNN**.<br>Value range of the GNN: 1 ... 199 |
| **2** | Enter the **port number** into register **235400 + GNN**.<br>Value range of GNN: 1 ... 199 |
| ⇨ | Now you can access the value via register **1nnnmmzzzz.**<br>Value range of GNN = nnn: 001 ... 199<br>Value range mm: 02 ... 17<br>Value range zzzz: 0000 ... 9999 |

This lets you directly access all JX3 module registers of the remote network node.

**Example**

Via network, a controller and a bus node JX3-BN-ETH are connected.
A JX3-AI4 module is connected to the bus node.

| Configuration of the bus node | Value |
|---|---|
| GNN | 3 |
| IP address | 192.168.10.14 |
| IP port | 50000 |

**Task:**

The trailing indicator of the analog channel 4 peak value is to be read.

**Solution:**

You create a JetSym STX program by taking the following steps:

- Register 235003 is loaded with the IP address of the bus node.
- Register 235403 is loaded with the IP port of the bus node.
- The value of network register 1003021421 is assigned to a local variable.

# Indirect addressing of remote modules

**Introduction**

Indirect addressing of network registers lets you access registers of a remote network node. First enter the number of the remote node register into a table of register numbers in the local controller. Content indexing of this table is carried out via the three low-order figures of the network register number.

**Registers - Overview**

Overview of the registers allowing indirect addressing of remote nodes:

| Register | Value range | Properties |
|---|---|---|
| 236000 + zzz | 236000 ... 236199 | Register table for the **register numbers** |
| 1nnn980zzz | 1nnn980000 ... 1nnn980199 | Register array for the **Content** |

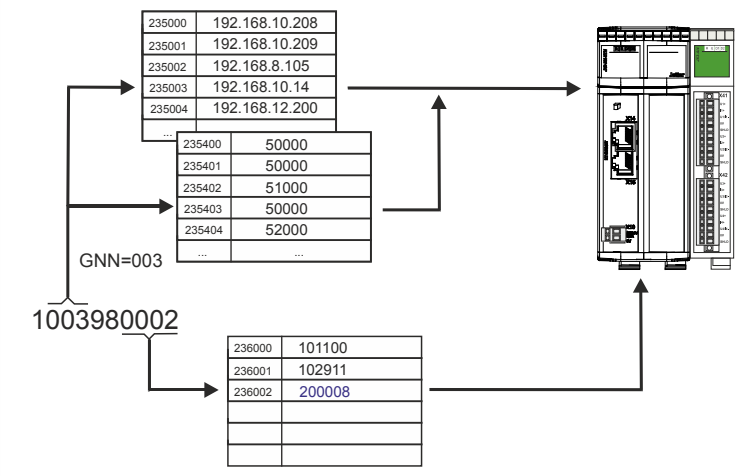
Note on the contents of the table:

- nnn = GNN in the range 000 ... 199
- zzz in the range 000 ... 199

**Indirect network register access**

For indirect access to a remote node via network register, the controller uses the first part of the register prefix as an index to the tables containing the IP addresses and IP ports. The values read out of these tables are used by the controller to address the bus node in the network. For further information on registers 235000 ff, please refer to **Network registers - IP address and IP port** (see page 248).

The module register number (no. 5 in the **addressing scheme** (see page 248)) is used by the controller as an index to a table of register numbers. The register number read out of this table is used by the controller to address the register in the bus node.



**Action**

If you want to access the register of a remote network node using register addresses as of 1 billion, proceed as follows:

| Step | Action |
|---|---|
| **1** | Enter the **IP address** of the remote network node into register **235000 + GNN**.<br>Value range of the GNN: 0 ... 199 |
| **2** | Enter the **port number** into register **235400 + GNN**.<br>Value range of the GNN: 0 ... 199 |
| **3** | Enter the required **register number** of the remote network node into register **236000 + zzz**. |
| ⇨ | Now you can access the value via register **1nnn980zzz.**<br>Value range of the GNN: nnn = 000 ... 199<br>Value range zzz: 000 ... 199 |

This configuration lets you indirectly access - via 200 controller registers - all module registers of the remote network node.

**Example**

Via network, a controller and a bus node JX3-BN-ETH are connected.

| Configuration of the bus node | Value |
|---|---|
| GNN | 3 |
| IP address | 192.168.10.14 |
| IP port | 50000 |

**Task:**

The global error register of the JX3-BN-ETH is to be read every second.

**Solution:**

- Register 235003 is loaded with the IP address of the bus node.
- Register 235403 is loaded with the IP port of the bus node.
- Register 236028 is loaded with the error register number 200008.

# Addressing with variable destination window

**Introduction**

Indirect addressing also allows for a variable destination window. You shift the register array of 10,000 registers of the remote network nodes by an offset by entering a value into R 272702 of the remote network nodes.

**Registers - Overview**

Overview of the registers allowing indirect addressing with variable destination window:

| Register | Value range | Properties |
|---|---|---|
| 1nnn99zzzz | 1nnn990000 ... 1nnn999999 | **Register content** of a remote network node;<br>The register is in the variable destination window which consists of 10,000 registers. |
| 272702<br>(of the remote node) | 0 ... 2,147,483,647 | Variable destination window:<br>The destination window is a register array of a remote network node.<br>This destination window is shifted by this **offset**. |

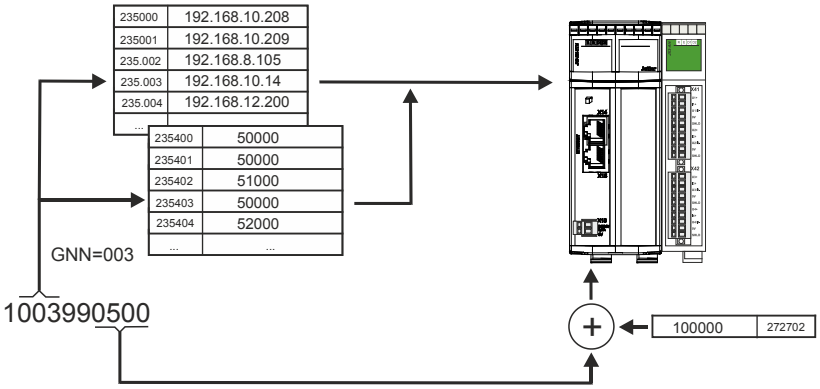
Note on the contents of the table:

- nnn = GNN in the range 000 ... 199
- zzzz in the range 0 ... 9,999

**Network register access with variable destination window**

For access via network register with variable destination window to a remote node, the controller uses the first part of the register prefix as an index to the tables containing the IP addresses and IP ports. The values read out of these tables are used by the controller to address the bus node in the network. For further information on registers 235000 ff, please refer to **Network registers - IP address and IP port** (see page 248).

The module register number (no. 5 in the **addressing scheme** (see page 248)) is used by the controller to address the register in the bus node. A register number is transmitted to the remote network node by the controller. The remote network node adds the content of register 272702 to this register number and uses the result as register number.



**Steps to take for addressing with destination window**

To use register addresses starting from 1 billion with variable destination window (offset), proceed as follows:

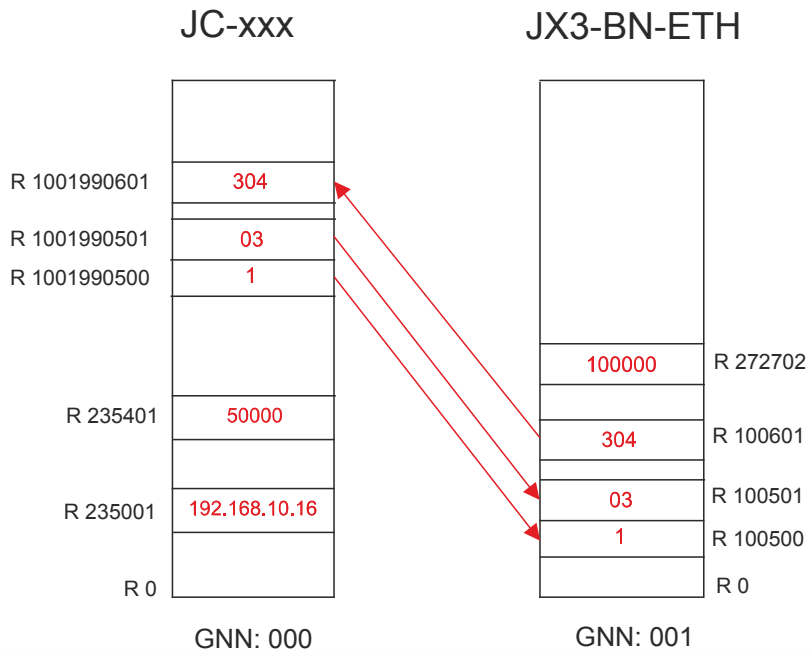
| Step | Action |
|---|---|
| **1** | Enter the **IP address** of the remote network node into register **235000 + GNN**.<br>Value range of the GNN: 0 ... 199 |
| **2** | Enter the **port number** into register **235400 + GNN**. Value range of the GNN: 0 ... 199 |
| **3** | Set the base address of the **destination window:** Enter a value into R 272702 of the remote network node. |
| ⇨ | Now, registers **1nnn990000 ... 1nnn999999** let you access the value. |

**Example**

A JetControl is to read a register value from a JX3-BN-ETH. Control system and bus node are interconnected via the Jetter Ethernet system bus.

There are JX3 modules connected to the JX3-BN-ETH, such as a JX3-AO4 of module number 03.

By entering value 100000 into R 272702 of the JX3-BN-ETH, you get read access to the EDS of the connected JX3 modules. In this example, the module code of the JX3-AO4 is to be read. For further information on how to read an EDS, please refer to **EDS registers** (see page 35).




Reading is carried out in three steps:

| Step | Action |
|---|---|
| **1** | Enter value 1 for a JX3 module into R 1001990500. |
| **2** | Enter module number 03 into R 1001990501. |
| **3** | Read module code 304 for JX3-AO4 from R 1001990601. |

# Register description - Acyclic data interchange

**Introduction**

In acyclic data interchange, data transmission from a controller to remote network nodes is carried out via JetIP protocol. The client in the controller is supplied with registers for configuration and error diagnostics.

**Registers/flags - Overview**

| Register | Description |
|---|---|
| **232708** | Timeout in milliseconds |
| **232709** | Response time in milliseconds |
| **232710** | Amount of network errors |
| **232711** | Error code of last access |
| **232717** | Maximum number of retries |
| **232718** | Present number of retries |

| Flags | Description |
|---|---|
| **2075** | Network error |

**R 232708**

**Timeout**

To R 232708, write the timeout (in milliseconds) for acyclic access via network.

| **Module register properties** | |
|---|---|
| Values | 1 ... 65,535 [ms] |
| Value after reset | 250 [ms] |

**R 232709**

**Response time**

R 232709 displays the total response time of latest acyclic access via network in milliseconds. The total response time includes the time for data transmission and the processing times in the controller and in the remote network node.

| **Module register properties** | |
|---|---|
| Values | 0 ... 65,535 [ms] |
| Type of access | Read |

**R 232710**

## Amount of network errors

R 232710 shows the total number of network errors.

| Module register properties | |
|---|---|
| Values | -2,147,483,648 ... 2,147,483,647 (overflowing) |

**R 232711**

## Error code

R 232711 shows the error code of the latest network access.

| Module register properties | | |
|---|---|---|
| Values | 0 | No errors |
| | 1 | Timeout |
| | 3 | Error message from remote node |
| | 5 | Invalid network address |
| | 6 | Invalid amount of registers |
| | 7 | Invalid interface number |

**R 232717**

## Maximum number of retries

R 232717 lets you set the maximum possible number of network access retries. If a network access could not be made without errors, the controller will repeat the access at the most as often as it has been set in this register. If the network access could still not be made without errors, the controller will cancel the access and create an error message.

| Module register properties | |
|---|---|
| Values | 0 ... 255 |

**R 232718**

## Present number of retries

R 232710 shows the total number of network access retries.

| Module register properties | |
|---|---|
| Values | -2,147,483,648 ... 2,147,483,647 (overflowing) |

**M 2075**

## Network error

If a network error occurs, the operating system sets flag 2075. In order to detect further errors this way, you must manually reset the flag.

| **Flag properties** | | |
|---|---|---|
| Values | 0 | No network errors since last reset |
| | 1 | A network error has occurred |

## 10.3.2 Cyclic data interchange

**Introduction**

This chapter covers cyclic data interchange via Jetter Ethernet system bus.

**Properties**

Properties of cyclic data interchange via Jetter Ethernet system bus:

| Property | Description |
|---|---|
| Architecture | Publish/subscribe<br>■ The publishers send the data.<br>■ The subscribers receive the data.<br>■ Usage of multicast frames |
| Publisher | ■ Each publisher sends one or several publications.<br>■ Data of a publication are consistently transferred in a frame.<br>■ The cycle time can be set for each publication. |
| Subscriber | ■ The subscriber receives one or several publications and assigns them to the corresponding subscriptions.<br>■ The subscriber validates the received data. |
| Data | ■ Registers<br>■ Inputs<br>■ Outputs |
| Access time | ■ Very short, as the network nodes access the locally stored interchanged data. |
| Configuration | ■ In the JetSym Hardware Manager |
| Reach | ■ Restricted to own subnet |

**Examples of the application**

- Cyclic, deterministic interchange of process data
- Cyclic, deterministic interchange of status information details

The JetSym Hardware Manager generates the configurations for cyclic data interchange using the status information details and the process data of the connected peripheral modules.

**Restrictions**

For cyclic data interchange, do not use any configuration registers or special registers. Access to these registers can take longer or trigger further action, which may lead to unwanted results.

**Multicast in other networks**

Please note that the Jetter Ethernet system bus operates with multicasts (multipoint connections). If you couple the Jetter Ethernet system bus with your local network, you have to filter out unwanted multicasts by a router.

As an alternative, the function **JetSync blockage** (see page 296) can be used, too.

**Technical specifications**

Technical specifications of cyclic data interchange via Jetter Ethernet system bus:

- Usage of multicast frames
- Reserved multicast groups: 255
- Multicast groups available to the user: 0 ... 254
- IP addresses for multicasts: 239.192.0.0 + multicast group
- MAC address for multicasts: 01:00:5E:40:00:00 + multicast group
- Maximum size of user data in a publication/subscription: 256 byte

**Contents**


| Topic | Page |
|---|---|
| Publish/subscribe | 261 |
| Publish/subscribe - Registers | 263 |
| Network registers, network inputs and outputs | 269 |

# Publish/subscribe

**Introduction**

Publish/subscribe is used as communication architecture for cyclic data interchange in the Jetter Ethernet system bus. The JetSym Hardware Manager generates the configurations for cyclic data interchange and transfers them to the controller. Based on this configuration, the configuration automatically carries out cyclic data interchange.

**Basic data interchange**

Basic data interchange via publish/subscribe is executed by the publishers and subscribers in the operating system of the Jetter devices at the Jetter Ethernet system bus.

**Publisher**

- The publishers publish data of the network node, on which they are being processed.
- A data record is published by the publisher. Therefore it is called **publication**.
- A publisher can manage several publications.

**Subscriber**

- The subscribers which are interested in these data receive the publications and transfer the contents to the data of the network node on which they are processed.
- A data record is received by the subscriber. Therefore it is called **subscription**.
- A subscriber can manage several subscriptions.
- To receive a publication, there must exist a corresponding subscription.
- One publication can be received by subscriptions on various network nodes simultaneously, as the publications are published via multicast frames.

**JetSym**

When a combination of a controller and one or several network nodes is configured in JetSym, the Hardware Manager generates the configuration files for the publishers and for the subscribers. The Hardware Manager generates one-to-one relationships between the publications and the subscriptions.

**Features of publish/subscribe**

If, in Hardware Manager, you add network nodes with the modules connected to them, Hardware Manager will automatically generate the process data belonging to these modules as publish/subscribe variables.

Features of publish/subscribe

| Parameter | Value | Description |
|---|---|---|
| Number of network nodes | 000 ... 199 | 200 network nodes max.:<br>They are entered into the Hardware Manager by their name and as GNN |
| Maximum amount of process variables per publication/subscription | 64 | 64 process variables max:<br>This corresponds to 256 bytes of process data |

| Parameter | Value | Description |
|---|---|---|
| Cycle time | 1 ... 2,147,483,647 ms | Default: 2 ms |

Network nodes are the controller, the communication modules and the bus nodes.

For details on characteristic features of publish/subscribe, please turn to chapter **Hardware Manager** (see page 272).

**Configuring and executing publish/subscribe**

Publish/subscribe is configured in the JetSym Hardware Manager. Publish/subscribe is executed by the operating system of the respective network node:

- Publishers and subscribers are configured by means of configuration files in the file system of the network nodes.
- The configuration file for the publisher is **/SysConfig/JetSync/Publisher.pub**.
- The configuration file for the subscriber is **/SysConfig/JetSync/Subscriber.sub**.
- Automatic restart of the publishers and subscribers takes place in a controller at each restart of the application program.
- In the other network nodes, automatic restart of the publishers and subscribers takes place during the booting phase.
- For applying publishers and subscribers in a controller, an application program must be executed with at least one task running.

For transferring the configuration, the Hardware-Manager takes the following steps:

| Step | Action |
|---|---|
| **1** | Stop all publishers and subscribers. |
| **2** | Transfer the configuration files to all network nodes. |
| **3** | Restart all publishers and subscribers. |

**Related topics**

- **Hardware Manager** (see page 272)

# Publish/subscribe - Registers

**Introduction**

If you transmit cyclic data by publish/subscribe, there are several module registers available for administration, configuration and error detection. You have got read and partial write access to these module registers.

**Register overview**

| Module registers | Description |
|---|---|
| **210004, 200008, 200009** | General error registers |
| **250000 ... 250004** | Registers for administration of all subscriptions |
| **250x10 ... 250x11** | Registers for administration of one subscription |
| **250x20 ... 250x30** | Registers for configuring one subscription |
| **254001 ... 254003** | Registers for error detection |
| **255000 ... 255004** | Registers for administration of all publications |
| **255x10 ... 255x11** | Registers for administration of one publication |
| **255x20 ... 255x30** | Registers for configuring one publication |
| **Flag 2080** | Enable for publishing an error |
| **Flag 2081** | Error collection of the subscriber |

x = 0 ... 9

**Availability**

Administration and configuration registers are available as follows:

- For subscriptions and publications, 10 arrays for administration and configuration registers are available.
- The register arrays differ by the hundred's place of the respective register number.
- The placeholder x indicates the number of the register array. Value range of x: 0 ... 9
- External clients use register array x = 1, such as JetSym with visualization application and PCOMX protocol.
- STX functions use register array x = 0.
- In order to gain faster access to individual publish/subscribe administration registers, several register arrays are at your disposal: There are individual publish/subscribe IDs to be called in each register array.

**Registers for administration of all subscriptions**

There are several registers available which go with all subscriptions.

| Register | Name | Description |
|---|---|---|
| **250000** | Status | Status register |
| **250001** | Command | Command register |
| **250002** | ID in case of error | Displays the ID of the subscription, in which an error has occurred. |
| **250003** | Amount | Total amount of subscriptions |
| **250004** | CRC | 16-bit CRC (**C**yclic **R**edundancy **C**ode) of the subscriber configuration file |

**Subscriber status**

**Status registers of all subscriptions**

From MR 250000, you can read the collective status of all subscriptions. In case of an error, you first read out the ID of the subscription, in which an error has occurred.

**Meaning of the individual bits**

**Bit 0** **Error in CRC computing of the configuration file**

0 = No error has occurred.

1 = For CRC computing, the configuration file does not exist. For this reason, CRC computing has not taken place.

**Bit 1** **Error in connection with a subscription**

1 = An error has occurred in a subscription.
At the moment, this is only a timeout error.

**Bit 7** **Subscription is functioning.**

0 = If a subscription fails, bit 7 is reset.

1 = The subscriptions are functioning.

**Module register properties**

Type of access Read

**Subscriber command**

**Command registers of all subscriptions**

Via MR 250001, you transmit commands to all subscriptions.

| Commands | |
|---|---|
| **102** | **Reboot all subscribers** |
| **105** | **Stop all subscribers** |
| **110** | **Acknowledge error** |

**Selecting a subscription**

The following registers let you select a subscription as follows:

- The index is for selecting subscriptions.
  - If the subscription exists, R 250x11 shows its ID.
  - If the subscription does not exist, R 250x11 shows value **-1**.
- In this case, enter the ID of the subscription into R 250x11.
  - If the subscription exists, the content of R 250x11 is kept.
  - If the subscription does not exist, R 250x11 shows value **-1**.

| Register | Name | Description |
|---|---|---|
| **250x10** | Index | Index of the subscriptions:<br>0: Selects the first subscription<br>1: Selects the next subscription<br>2: etc. |
| **250x11** | ID | The ID of the subscription is entered |

**Configuring a subscription**

The following registers show the configuration of a subscription, which you have selected via R 250x10 and R 250x11.

| Register | Name | Description |
|---|---|---|
| **250x20** | Status | Bit 0: Publication received<br>Bit 1: Timeout |
| **250x21** | Mode | 0: Cyclic<br>1: Upon request |
| **250x22** | Number of variables | As configured |
| **250x23** | Group address | As configured |
| **250x24** | Hash | Internal usage |
| **250x25** | Sequence number | Internal usage |
| **250x26** | Data size | Internal usage |
| **250x27** | Timeout in ms | Bus cycle * 3 |

| Register | Name | Description |
|---|---|---|
| **250x28** | Number of received publications | - |
| **250x29** | Amount of timeouts | - |
| **250x30** | Amount of missing sequence numbers | The subscriber of a publication computes the difference between present and last received sequence number. If the value of the difference is greater than one, certain publications have not been received. |

**Registers for error detection**

If a subscription has not received any process data from the assigned publication before timeout, the subscription will generate an error. Further, the operating system writes the address of the bus node into registers 254001 to 254003, with which communication has been terminated.

This helps you to search for the error exactly in this bus node using NetCopy commands.

| Register | Name | Description |
|---|---|---|
| **254001** | GNN | Global Node Number |
| **254002** | IP address | |
| **254003** | Port number | |

**Registers for administration of all publications**

There are several registers available which go with all publications.

| Register | Name | Description |
|---|---|---|
| **255000** | Status | Status register |
| **255001** | Command | Command register |
| **255002** | ID in case of error | Displays the ID of the publication, in which an error has occurred. |
| **255003** | Amount | Amount of all publications |
| **255004** | CRC | 16-bit CRC (Cyclic Redundancy Code) of the publication configuration file |

**Publisher status**

**Status registers of all publications**

From MR 255000, you can read the collective status of all publications. In case of an error, you first read out the ID of the publication, in which an error has occurred.

**Meaning of the individual bits**

**Bit 0** **Error in CRC computing of the configuration file**

0 = No error has occurred.

1 = For CRC computing, the configuration file does not exist. For this reason, CRC computing has not taken place.

| | |
|---|---|
| **Bit 1** | **Error in connection with a publication** |
| | 1 = An error has occurred in a publication. |
| **Bit 7** | **Publication is functioning** |
| | 0 = If a publication fails, bit 7 is reset. |
| | 1 = The publications are functioning. |

| **Module register properties** | |
|---|---|
| Type of access | Read |

**Publisher command**

**Command registers of all publications**

Via MR 255001, you transmit commands to all publications.

| **Commands** | |
|---|---|
| **102** | **Reboot all publishers** |
| **105** | **Stop all publishers** |
| **110** | **Acknowledge error** |

**Selecting a publication**

The following registers let you select a publication:

- The index is for selecting publications.
  - If the publication exists, R 255x11 shows its ID.
  - If the publication does not exist, R 255x11 shows value **-1**.
- In this case, enter the ID of the publication into R 255x11.
  - If the publication exists, the content of R 255x11 is kept.
  - If the publication does not exist, R 255x11 shows value **-1**.

| Register | Name | Description |
|---|---|---|
| **255x10** | Index | Index of the publications:<br>0: Selects the first publication<br>1: Selects the next publication<br>2: etc. |
| **255x11** | ID | The ID of the publication is entered |

**Configuring a publication**

The following registers show the configuration of a publication, which you have selected via R 255x10 and R 255x11.

| Register | Name | Description |
|---|---|---|
| **255x20** | Status | Bit 0: Publication transmitted |
| **255x21** | Mode | 0: Cyclic<br>1: Upon request |

| Register | Name | Description |
|---|---|---|
| **255x22** | Number of variables | As configured |
| **255x23** | Group address | As configured |
| **255x24** | Hash | Internal usage |
| **255x25** | Sequence number | Internal usage |
| **255x26** | Data size | Internal usage |
| **255x27** | Timeout in ms | Bus cycle |
| **255x28** | Number of publications sent | - |
| **255x29** | Number of retries | - |
| **255x30** | Number of transmit errors | - |

# Network registers, network inputs and outputs

**Introduction**

The network registers, network inputs and outputs let you access in transparent mode, at cyclic data interchange, registers, inputs and outputs of remote nodes. The controller accesses the local image of the cyclic data.

**Prerequisites**

These are the prerequisites for using the registers, inputs and outputs at cyclic data interchange:

- Via publish/subscribe, the data are interchanged in cyclic mode.

**Properties**

Network registers, network inputs and outputs are not used in cyclic data interchange:

- If network registers of non-cyclic data interchange are accessed, the controller generates acyclic network register access.
- If network inputs and outputs of non-cyclic data interchange are accessed, the controller does not generate acyclic network register access. There are no data being transmitted via network.

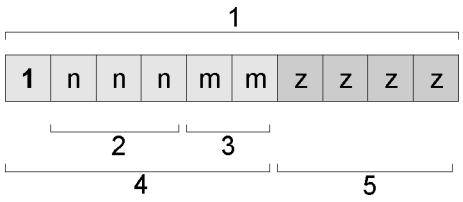
**Advantages of network registers, network inputs and outputs**

Advantages of network registers, network inputs and outputs in cyclical data interchange as compared with acyclic data interchange:

- The operating system cyclically interchanges data of the registers, inputs and outputs with other network nodes.
- This results in network load optimization.
- This is a very quick access, as, at the instance of use, only the local images of the data have to be accessed.

**Register addressing scheme**

The addressing scheme for network registers is as follows:




| No. | Element | Description |
|---|---|---|
| **1** | Register number | Supports direct access |
| **2** | First part of register prefix: Bus node ID, GNN | nnn = 001 ... 199: ID of the network node, referred to as Global Node Number |
| **3** | Second part of register prefix: Number of the function module | mm = 02 ... 17: Number of the JX3 module of a remote node<br>mm = 91: Registers of the combined digital inputs and outputs of a remote node |
| **4** | Part 1 + 2: Register prefix | 1nnnmm: The prefix is preceded by a leading ONE. |

| No. | Element | Description |
|---|---|---|
| **5** | Module register number | zzzz = 0000 ... 9999 |

**Network registers for accessing JX3 modules**

Characteristic feature of the register number for access to remote JX3 modules: The value of the second part of the register prefix is the number of the module at the JX3 system bus (02 ... 17).

In cyclic data interchange, access to the process data of the remote JX3 modules is made via network registers.

For further information on configuration of data interchange and generated variables for access to JX3 modules, please turn to chapter **Hardware Manager** (see page 272).

**Register overview - Inputs and outputs**

The register number, in which the digital inputs and outputs of the remote nodes have been combined, is characterized by the value being 91 in the second part of the register prefix.

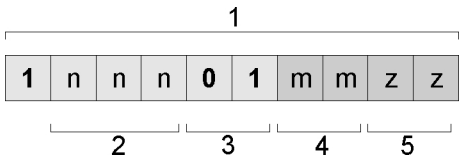
**Overview**

| Registers | Description |
|---|---|
| **1nnn914000 ... 1nnn914030** | 32 combined inputs |
| **1nnn914060 ... 1nnn914092** | 16 combined inputs |
| **1nnn914120 ... 1nnn914153** | 8 combined inputs |
| **1nnn914200 ... 1nnn914230** | 32 combined outputs |
| **1nnn914260 ... 1nnn914292** | 16 combined outputs |
| **1nnn914320 ... 1nnn914353** | 8 combined outputs |

Where nnn = GNN: 000 ... 199

**Addressing scheme - Inputs and outputs**

The addressing scheme for the digital network inputs and outputs at cyclic data interchange is as follows:




| No. | Element | Description |
|---|---|---|
| **1** | I/O number | Supports direct access |
| **2** | Bus node ID, GNN | nnn = 001 ... 199: ID of the network node, referred to as Global Node Number. |
| **3** | Designation:**01**:<br>I/O 01 as a fixed number | 01: 01 indicates that a JX3 module is to be addressed. |
| **4** | Module number | mm = 02 ... 17: Number of the JX3 module of a remote node |
| **5** | Module-specific I/O number | zz = 01 ... 16: Specifies which input/output on the module is to be addressed |

**Example**

Via network, a controller and a bus node JX3-BN-ETH are connected. A JX3-DO16 is connected to the bus node. The JX3-DO16 has got I/O module number 3.

**Task:**

The outputs of the JX3-DO16 are to be activated or deactivated as follows:

| Step | Description |
|---|---|
| **1** | All outputs with odd numbers are active for half a second, while all outputs with even numbers are deactivated. |
| **2** | All outputs with even numbers are active for half a second, while all outputs with odd numbers are deactivated. |
| **3** | There is a moving light from output 1 to output 16; each corresponding output is activated for 200 ms. |
| **4** | Proceed with step 1. |

**Solution:**

Configure the network group in the JetSym Hardware Manager and write an application program. Download both to the network nodes.

**Related topics**

- **Hardware Manager** (see page 272)

## 10.3.3 Hardware Manager

**Introduction**

The Hardware Manager lets you easily configure the peripheral devices. If possible, always use the Hardware Manager that is part of JetSym. Making configurations by hand is complicated and prone to errors

**Detailed information**

For detailed information on hardware configuration using Hardware Manager, refer to the JetSym help.

**Contents**


| Topic | Page |
|---|---|
| Hardware Manager | 273 |

# Hardware Manager

**Hardware Manager**

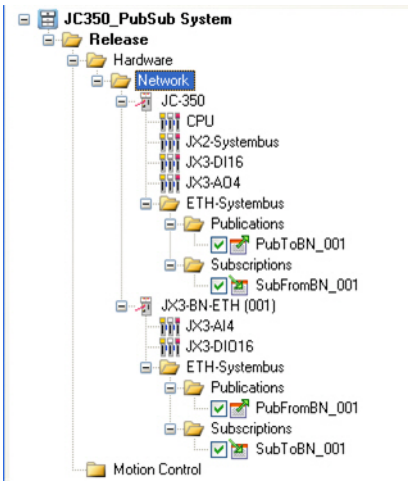
The Hardware Manager manages all connected hardware components.

The Hardware Manager assists you in the following aspects:

- Engineering and configuring control systems and bus nodes
- Engineering modules and axes at the JX2 system bus and configuring axes at the JX2 system bus
- Engineering JX3 modules at a JX3-BN-ETH and a JC-3xx
- Engineering and configuring Ethernet axes
- Engineering an axis group (path group and technology group)
- Configuring a path group
- Configuring technology group

**Launching the Hardware Manager**

For launching the Hardware Manager, klick, in JetSym, the tab **Hardware**. As an alternative, launch the Hardware Manager via keys **[Alt] + [5]**.




**Related topics**

- **Ethernet system bus** (see page 240)

## 10.3.4 Error handling at the Jetter Ethernet system bus

**Introduction**

This chapter covers error handling at the Jetter Ethernet system bus.

**Contents**


| Topic | Page |
|---|---|
| Acyclic data interchange - Error logging | 275 |
| Error message during CRC computing | 276 |
| Error message on part of a subscription | 277 |
| Controller evaluates errors reported by a remote network node | 278 |

# Acyclic data interchange - Error logging

**Introduction**

The programmer uses the following information for error logging:

- Return values of the commands
- JetIP networking registers and flags

**NetCopy() and NetBit()**

For error logging, use the return values of the respective command. You will find them in the JetSym online help.

Jetter AG recommends not to execute error logging via the registers and flags of the JetIP network.

**Network registers**

Error logging for network registers and flags of the JetIP networking:

| Registers/flags | Description |
|---|---|
| Flag 2075 | Errors at acyclic data interchange |
| Register 232710 | Amount of errors at acyclic data interchange |
| Register 232711 | Error code of the latest acyclic data interchange |

## Error message during CRC computing

**Detecting the error**

Both publisher and subscriber carry out a CRC of their configuration files. The calculated value can be read from registers 255004 and 250004. If there is no configuration file, they report an error.

**Root cause of the error**

This error may be caused by the following root cause:

- CRC computing failed, because there is no configuration file.

**Response of the device to this error**

The operating system of the device responds to the error by taking the following steps:

| Step | Description |
|---|---|
| 1 | The operating system sets bit 0 in the status register of the publisher (R 255000) or of the subscriber (R 250000). |

**Fixing the root cause**

Deploying a configuration file

**Acknowledging the error**

After deploying a configuration file, restart both publisher and subscriber.

## Error message on part of a subscription

**Detecting the error**

If a subscriber has not received any process data from the assigned publisher before timeout, the subscriber will generate an error. The subscriber for the subscription of which the error has been generated, can run either on a controller or on a network node. The remote network node is a JX3-BN-ETH, for example.

**Root cause of the error**

The error may be caused as follows:

- Communication with the network client providing the process data is terminated.

**Response of the device to this error**

The operating system of the device responds to the error by taking the following steps:

| Step | Description | |
|---|---|---|
| 1 | Sets bit 1 in R 250000. | |
| 2 | Writes the subscription ID to R 250002. | |
| 3 | Sets flag 2081. | |
| 4 | Writes value 11103 and the ID to the error buffers.<br>The error buffer can be accessed via registers 380000 ff. (error history). | |
| 5 | Writes the GNN of the network node communication with which has been terminated to R 254001. | |
| 6 | Writes the IP address of the network node communication with which has been terminated to R 254002. | |
| 7 | Writes the port number of the network node communication with which has been terminated to R 254003. | |
| 8 | **If ...** | **... then ...** |
| | ... flag 2080 is set, | ... bit 3 is set in R 210004 and R 200008. The red status LED of the controller is lit. |

**Fixing the root cause**

By means of NetCopy commands, you can precisely locate the error and fix the root cause. This works, because GNN, IP address and port number of the other network node are known.

**Acknowledging the error**

To acknowledge the error, write command 110 to register 250001.

## Controller evaluates errors reported by a remote network node

**Access to the status registers**

The controller has got read access to the contents of the following status registers of all network nodes at the Jetter Ethernet system bus.

The contents are accessed via registers 39nnn0 through 39nnn5. (GNN: nnn = 001 ... 199).

| Registers | JX3-BN-ETH, JX3-COM-EIPA | Controller |
|---|---|---|
| Error register | 200008 | 39nnn0 |
| Enhanced error register 1 | 200009 | 39nnn1 |
| Enhanced error register 2 | 200010 | 39nnn2 |
| JetSync status | 240010 | 39nnn3 |
| Subscriber status | 250000 | 39nnn4 |
| Subscription ID | 250002 | 39nnn5 |

The operating system writes the ID of the subscription for which last an error has been reported to register 250002.

**Locating faults**

If the value of register 39nnn0 is unequal zero, an error has occurred. A network node has reported this error to the controller via its status registers.

In consequence, the operating system of the controller reacts by taking the following steps:

| Step | Description | | |
|---|---|---|---|
| 1 | The operating system sets bit 10 in R 200009. | | |
| 2 | **If ...** | **... or ...** | **... then ...** |
| | ... Bit x = 1 of R 200009, | Bit x = 1 of R 200010, | ... the operating system sets bit 7 of R 200008. |
| 3 | The operating system enters the GNN of the network node having last reported an error to the controller into R 394001. | | |
| 4 | The operating system enters the IP address of the network node having last reported an error to the controller into R 394002. | | |
| 5 | The operating system enters the port number of the network node having last reported an error to the controller into R 394003. | | |

**Fixing the root cause**

By means of NetCopy commands, you can precisely locate the error and fix the root cause. This works, because GNN, IP address and port number of the other network node are known.

Make sure the contents of registers 39nnn0 through 39nnn5 are read by the application program. Further registers having got a value unequal zero indicate that further network nodes have reported an error. Make sure you also clear these errors.

# 10.3.5 JetIPScan - Register description

**Introduction**

This chapter describes the registers from which the status information of the JetIPScan feature can be read out. You can use these registers for debugging or diagnostics. Further features, such as, for example, checking the network configuration, cannot be triggered this way.

**Contents**


| Topic | Page |
|---|---|
| Register numbers | 280 |
| Global status - Register description | 281 |
| Warnings and errors - Register description | 284 |
| Configuration - Register description | 288 |

# Register numbers

**Introduction**

Status information is displayed within the registers of a coherent register block. The basic register number of this block is dependent on the controller.

**Register numbers**

| Device | Basic register number | Register numbers |
|---|---|---|
| JC-360 | 520000 | 520000 ... 522999 |

**Determining the register number**

In this chapter, only the last four figures of a register number are specified. e.g. MR 1499. Add to this module register number the basic register number of the corresponding device to determine the complete register number, for example 521499.

**Registers - Overview**

| Register | Description |
|---|---|
| **MR 0 ... MR 13** | Global status |
| **MR 1000 ... MR 1499** | Warnings and errors |
| **MR 2000 ... MR 2399** | SET and ACTUAL configurations |

# Global status - Register description

**Introduction**

The current I/O size can be read from this register.

**MR 0**

**State of the total**

In MR 0. the controller signals a summary of status messages in bit-coded mode.

| Meaning of the individual bits | | |
|---|---|---|
| **Bit 0** | **Function enable** | |
| | This bit corresponds to bit 2 of the system status register 202962. | |
| | 0 = | JetIPScan client - OFF |
| | 1 = | JetIPScan client - ON |
| **Bit 1** | **Collective error message** | |
| | 1 = | Reg 13 contains value 0 |

| Module register properties | |
|---|---|
| Type of access | Read |
| Value after reset | Bit 0: Depends on release status.<br>Bit 1: 0 |

**MR 10**

**State of execution**

Corresponds to the feedback value *State*.

| Module register properties | | |
|---|---|---|
| Values | 0 | The function is not active.<br>Function terminated. |
| | 1 | Waiting for response from network nodes |
| | 2 | Send an inquiry telegram |
| | 3 | Check the replies sent by the nodes |
| | 4 | Write the configurations of the nodes |
| Type of access | Read | |

**MR 11** **Number of cycles**

Corresponds to the feedback value *Count*.

| Module register properties | | |
|---|---|---|
| Values | 0 ... 3 | Number of cycles |
| Type of access | Read | |

**MR 12** **Number of changes**

Corresponds to the feedback value *Changed*.

| Module register properties | | |
|---|---|---|
| Values | 0 ... 199 | Number of changed network nodes |
| Type of access | Read | |

**MR 13** **Result of the function**

Corresponds to the feedback value *Result* and the register content of the global error number 2000051. This register indicates the value of the latest error or warning. Values greater than zero indicate warnings. Values smaller than zero are error messages.

| Module register properties | | |
|---|---|---|
| Values | 0 | No error or warning |
| | 5 | The user has terminated the function |
| | 1001 | The first received response does not match response 2 and 3 (see MR 101x) |
| | 1002 | The second received response does not match response 1 and 3 (see MR 102x) |
| | 1003 | The third received response does not match response 1 and 2 (see MR 103x) |
| | -1 | All 3 responses are dissimilar (see MR 100x) |
| | -2 | The IP settings of at least one node could not be changed (see MR 140x) |
| | -3 | The JetIPScan function has been invoked, although it is active already |
| | -10 | The length of the set value list is < 1 or > 255, or the pointer to the list is invalid |
| | -11 | A GNN of the set value list is < 1 or > 255, or it is a multiple GNN |
| Values | -20 ... -40 | Internal error |

**Module register properties**

| | | |
|---|---|---|
| | -1001 ... -1199 | The node has reported the wrong CtrlID or CtrlIDopt (see MR 110x) |
| | -2001 ... -2199 | The node has not called (see MR 120x) |
| | -3001 ... -3199 | Several nodes of the same GNN have called (see MR 130x) |
| Type of access | Read | |

# Warnings and errors - Register description

**Introduction**

Detailed diagnostics of the warnings and errors which have occurred can be carried out by means of these registers.

If, during checking and setting the IP address of all nodes a warning or an error occurs, the controller sets the corresponding bit in the registers described below. In this case, the bit corresponds to the GNN of the node.

The GNN of the node and the bit number relate as follows:

Bit number = GNN - 1

As a register contains 32 bit, individual groups of 7 subsequent registers each are created (see table).

| Register bit | GNN |
|---|---|
| Register.0 | 1 |
| Register.31 | 32 |
| (Register + 1).0 | 33 |
| (Register + 1).31 | 64 |
| (Register + 2).0 | 65 |
| (Register + 2).31 | 96 |
| (Register + 3).0 | 97 |
| (Register + 3).31 | 128 |
| (Register + 4).0 | 129 |
| (Register + 4).31 | 160 |
| (Register + 5).0 | 161 |
| (Register + 5).31 | 192 |
| (Register + 6).0 | 193 |
| (Register + 6).6 | 199 |

**MR 1000 ... 1006**

**All 3 responses are dissimilar**

The controller scans the network configuration three times and compares the three replies. If all three replies are dissimilar, the controller sets the respective bit in these registers.

| **Meaning of the individual bits** | |
|---|---|
| Bit = 0 | No error |
| Bit = 1 | Error |

| **Module register properties** | |
|---|---|
| Bit number | GNN - 1 |

| | |
|---|---|
| Type of access | Read |

**MR 1010 ... 1016**

## Reply no. 1 is not the same as replies 2 and 3

The controller scans the network configuration three times and compares the three replies. If replies 2 and 3 are the same, yet reply 1 is different, the controller sets the respective bit in these registers.

| **Meaning of the individual bits** | |
|---|---|
| Bit = 0 | No warning |
| Bit = 1 | Warning |

| **Module register properties** | |
|---|---|
| Bit number | GNN - 1 |
| Type of access | Read |

**MR 1020 ... 1026**

## Reply no. 2 is not the same as replies 2 and 3

The controller scans the network configuration three times and compares the three replies. If replies 1 and 3 are the same, yet reply 2 is different, the controller sets the respective bit in these registers.

| **Meaning of the individual bits** | |
|---|---|
| Bit = 0 | No warning |
| Bit = 1 | Warning |

| **Module register properties** | |
|---|---|
| Bit number | GNN - 1 |
| Type of access | Read |

**MR 1030 ... 1036**

## Reply no. 3 is not the same as replies 2 and 3

The controller scans the network configuration three times and compares the three replies. If replies 1 and 2 are the same, yet reply 3 is different, the controller sets the respective bit in these registers.

| Meaning of the individual bits | |
|---|---|
| Bit = 0 | No warning |
| Bit = 1 | Warning |

| Module register properties | |
|---|---|
| Bit number | GNN - 1 |
| Type of access | Read |

**MR 1100 ... 1106**

## Wrong CtrlID or CtrlIDopt

A node having got the required GNN has called, yet, the CtrlID or CTRLIDopt do not agree with it.

| Meaning of the individual bits | |
|---|---|
| Bit = 0 | No error |
| Bit = 1 | Error |

| Module register properties | |
|---|---|
| Bit number | GNN - 1 |
| Type of access | Read |

**MR 1200 ... 1206**

## The node has not called

The node having got the required GNN has not called.

| Meaning of the individual bits | |
|---|---|
| Bit = 0 | No error |
| Bit = 1 | Error |

| Module register properties | |
|---|---|
| Bit number | GNN - 1 |
| Type of access | Read |

**MR 1300 ... 1306**

## Multiple call

Several nodes using the same GNN have called. Yet, each node must have a unique GNN.

| Meaning of the individual bits | |
|---|---|
| Bit = 0 | No error |
| Bit = 1 | Error |

| Module register properties | |
|---|---|
| Bit number | GNN - 1 |
| Type of access | Read |

**MR 1400 ... 1406**

## The IP settings could not be changed

When the IP settings of a node have been changed, the controller checks whether the node has taken over these changes.

If the node has not taken over these changes, the controller sets the respective bit in these registers.

| Meaning of the individual bits | |
|---|---|
| Bit = 0 | No error |
| Bit = 1 | Error |

| Module register properties | |
|---|---|
| Bit number | GNN - 1 |
| Type of access | Read |

# Configuration - Register description

**Introduction**

These registers can be used to check the SET configuration and the three received ACTUAL configurations When you have entered the GNN in MR 2000, the controller transfers the values to the 4 register arrays.

**MR 2000**

## GNN

Enter the GNN here.

| Module register properties | |
|---|---|
| Values | 1 ... 199 |
| Value after reset | 1 |

**MR 2010 ... 2015**

## SET configuration

These registers let you read the default SET configuration.

| Register | Command line parameter |
|---|---|
| 2010 | NodeID (GNN) |
| 2011 | CtrlID |
| 2012 | CtrlIDopt |
| 2013 | IpAddr |
| 2014 | IpMask |
| 2015 | Gateway |

**MR 2110 ... 2123**

## ACTUAL configuration 1

These registers let you read the first received ACTUAL configuration.

| Register | Command line parameter |
|---|---|
| 2110 | NodeID (GNN) |
| 2111 | CtrlID |
| 2112 | CtrlIDopt |
| 2113 | IpAddr |
| 2114 | IpMask |
| 2115 | Gateway |
| 2120 | Quantity |
| 2121 | MAC address high |
| 2122 | MAC address low |

| Register | Command line parameter |
|---|---|
| 2123 | Sent IP address |

**MR 2210 ... 2223**

## ACTUAL configuration 2

These registers let you read the second received ACTUAL configuration.

| Register | Command line parameter |
|---|---|
| 2210 | NodeID (GNN) |
| 2211 | CtrlID |
| 2212 | CtrlIDopt |
| 2213 | IpAddr |
| 2214 | IpMask |
| 2215 | Gateway |
| 2220 | Quantity |
| 2221 | MAC address high |
| 2222 | MAC address low |
| 2223 | Sent IP address |

**MR 2310 ... 2323**

## ACTUAL configuration 3

These registers let you read the third received ACTUAL configuration.

| Register | Command line parameter |
|---|---|
| 2310 | NodeID (GNN) |
| 2311 | CtrlID |
| 2312 | CtrlIDopt |
| 2313 | IpAddr |
| 2314 | IpMask |
| 2315 | Gateway |
| 2320 | Quantity |
| 2321 | MAC address high |
| 2322 | MAC address low |
| 2323 | Sent IP address |

# 10.3.6 Administrating the connections of the JetIP/TCP and STX debug server

**Introduction**

This document covers the connection management enhancements of the JetIP/TCP server and of the STX debug server in a JetControl PLC.

If, for example, the Ethernet cable was unplugged or cut, the node was not able to clear the connection. The connection remained active.

The enhanced connection management allows for the server to clear connections according to criteria that can be set by the user.

**Number of connections**

The number of simultaneously established connections for the TCP server in a JetControl is limited to the following value:

| Server | Connections |
|---|---|
| JetIP/TCP server | 4 |
| STX debug server | 20 |

**Contents**


| Topic | Page |
|---|---|
| Automatic termination of connections | 291 |
| Register | 293 |

# Automatic termination of connections

**Introduction**

If the maximum number of simultaneously established connections has been reached, any further connections cannot be established. If further connect requests are made, the user can set the response by the JetIP/TCP server and of the STX Debug server. There are the following possibilities:

- Reject new connection.
- Terminate one existing connection and establish the new one.
- Terminate all existing connections and establish the new one.

**Default setting**

By default, the server terminates the connection with the longest time of inactivity.

**No automatic termination of connections**

If the server is not to terminate any of the existing connections, proceed as follows:

| Step | Action |
|---|---|
| **1** | Enter value 0 into MR 1. |

**Terminating the connection with the longest time of inactivity**

If the server is to terminate the connection that has been inactive the longest time, proceed as follows:

| Step | Action |
|---|---|
| **1** | Enter value -1 into MR 2. |
| **2** | Enter value 1 into MR 1. |

**Terminating the connection when the set minimum time has expired**

If the server is to terminate a connection after a set minimum time of inactivity, proceed as follows:

| Step | Action |
|---|---|
| **1** | Enter the minimum time [ms] into MR 2. |
| **2** | Enter value 1 into MR 1. |

If the set minimum value has not been exceeded yet, the server rejects the new connection.

**Terminating any connection**

If the server is to terminate any of the existing connections, proceed as follows:

| Step | Action |
|---|---|
| **1** | Enter value 2 into MR -1. |
| **2** | Enter value 1 into MR 2. |

**Terminating all connections which exceed the minimum time of inactivity**

If the server is to terminate all existing connections which have exceeded the minimum time of inactivity proceed as follows:

| Step | Action |
|---|---|
| **1** | Enter the minimum time [ms] into MR 2. |
| **2** | Enter value 1 into MR 2. |

# Register

**Register numbers**

The register numbers to be used are calculated by adding and the controller-dependent basic register number and the module register number.

| Controller/server | Basic register number | Register numbers |
|---|---|---|
| JC-360: JetIP/TCP | 230000 | 230000 ... 230002 |
| JC-360: STX-Debug | 212000 | 212000 ... 212002 |

**MR 0**

## Number of connections

The number of currently established connections can be read from module register 0.

| Module register properties | |
|---|---|
| Values | 0 ... 4 (JetIP/TCP server)<br>0 ... 20 (STX debug server) |

**MR 1**

## Mode

If the maximum number of connections is active, and if the server is to establish a new connection, module registers 1 and 2 determine the behavior.

| Module register properties | |
|---|---|
| Values | 0 ... 2 |
| Value after reset | 1 |

**MR 2**

## Minimum inactivity time

If the maximum number of connections is active, and if the server is to establish a new connection, module registers 1 and 2 determine the behavior.

| Module register properties | |
|---|---|
| Values | -1 ... 2,147,483,647 [ms] |
| Value after reset | -1 |

## 10.3.7 Executing an ARP request

**Use case**

Several controllers are interconnected via the Jetter Ethernet system bus. This is the case now. Controller B is exchanged. In this case, the IP address remains the same, but the Ethernet address (MAC address) changes. This way, data interchange between controller A and the new controller B is not possible.

To enable data interchange between the two controllers again, controller A would have to be relaunched.

To prevent a relaunch of controller A, an ARP request must be executed on controller A.

**Phases of an ARP request**

The controller A inquires from the Jetter Ethernet system bus, which node has got which specific IP address. Controller B reports that it has got this IP address. MAC address and IP address of controller B are aligned with each other. Now, controller A is informed of the MAC address which controller B has got. From now on, data interchange is possible again.

**Contents**


| Topic | Page |
|---|---|
| Executing an ARP request | 295 |

# Executing an ARP request

**ARP request**

When you enter the IP address of a network node into the corresponding register, the controller triggers an ARP request. This request is used for resolution of an IP address into an Ethernet address (MAC address).

**R 104250** **Executing an ARP request**

| Register properties | |
|---|---|
| Values | Valid IP address |

## 10.3.8 JetSync blockage

**Introduction**

In this chapter, the system command registers and the system commands for activating and deactivating the JetSync blockate will be explained in detail.

**Contents**


| Topic | Page |
|---|---|
| Description of system command registers | 297 |
| Description of the JetSync blockage system commands | 300 |

# Description of system command registers

**Registers - Overview**

The following registers are used in this manual:

| Registers | Description |
|---|---|
| **R 202960** | System password register |
| **R 202961** | System command register |
| **R 202962** | System status register |

**R 202960**

## System password register

Enter system password 1112502132 (0x424F6F74) into this register. Then enter the required command value into the system command register. Now, the controller sets the value of this register to 0.

| **Register properties** | |
|---|---|
| Value | 1112502132 (0x424F6F74) |

**R 202961**

## System command register

Enter the system commands into this register. Then the controller executes the command. Then, it sets the value of this register to 0.

| **Commands** | |
|---|---|
| **102** | **Restart the controller** |
| **104** | **Reset remanent parameters** |
| **122** | **Deactivate - Wait for communication** |
| **123** | **Activate - Wait for communication** |
| **160** | **Deactivate - Task switch on I/O access** |
| **161** | **Activate - Task switch on I/O access** |
| **170** | **Deactivate - Resume task time slot** |

| Commands | |
|---|---|
| **171** | **Activate - Resume task time slot** |
| **310** | **Load configuration files** |
| **311** | **Load module configuration** |
| **312** | **Load process data configuration for Ethernet system bus** |
| **313** | **Stop process data communication for Ethernet system bus** |
| **330** | **Disable JetIPScan client** |
| **331** | **Enable JetIPScan client** |
| **410** | **Disable JetSync blockage** |
| **411** | **Enable JetSync blockage for all ports** |
| **412** | **Enable JetSync blockage for port X15** |

| Register properties | |
|---|---|
| Access | System password register contains the correct password. |

**R 202962**

## System status register

The system status register lets you evaluate the system conditions.

| Meaning of the individual bits | | |
|---|---|---|
| **Bit 0** | **Task switch on I/O access** | |
| | 0 = | No task switching in the application program on I/O access. |
| | 1 = | Task switching is carried out in the application program on I/O access. |
| **Bit 1** | **Wait for communication** | |
| | 0 = | The controller waits for communication requests for a short time. |
| | 1 = | The controller does not wait for communication requests. |

| Meaning of the individual bits | | |
|---|---|---|
| **Bit 2** | **JetIPScan client** | |
| | 0 = | JetIPScan client not active |
| | 1 = | JetIPScan client active |
| **Bit 3** | **Resume task time slice** | |
| | 0 = | When a normal application task has been interrupted by a cyclic task or the Ethernet system bus publisher, the following application task is processed. |
| | 1 = | When a normal application task has been interrupted by a cyclic task or the Ethernet system bus publisher, processing the interrupted application task is resumed. |
| **Bit 8** | **JetSync blockage** | |
| | 0 = | JetSync blockage is not active |
| | 1 = | JetSync blockage is active |

| Register properties | |
|---|---|
| Access | Read |

# Description of the JetSync blockage system commands

**System command 410**

**Disable JetSync blockage**

**Effect:**

- The JetSync blockage is disabled for all ports. Bit 8 in R 202962 is reset.
- The Jetter Ethernet system bus multicast frames are transmitted to all ports (X14, X15 and CPU).

**Purpose:**

The JetSync blockage enabled by system command 411 or 412 is disabled. Forwarding the Jetter Ethernet system bus multicast frames to all ports again corresponds to the on-state of the controller.

**System command 411**

**Enable JetSync blockage for all ports**

**Effect:**

- The JetSync blockage is enabled for all ports (X14, X15, and CPU). Bit 8 in R 202962 is set.
- Jetter Ethernet system bus multicast frames which are received on a certain port are not forwarded to any of the other ports.
- All other Ethernet frames are forwarded as usual.

**Purpose:**

This command lets you prevent forwarding Jetter Ethernet system bus multicast frames to the CPU and the other ports. This way, networks are split and thus data traffic - e.g. from the machine network to higher-level networks - is reduced.

**Address space**

Splitting is carried out on Ethernet level via the multicast address range of the Jetter Ethernet system bus.
0x01 00 5E 40 00 00 ... 0x01 00 5E 40 00 FF

**System command 412**

**Enable JetSync blockage for port X15**

**Effect:**

- The JetSync blockage is enabled for port X15 only. Bit 8 in R 202962 is set.
- Jetter Ethernet system bus multicast frames of the CPU are forwarded to port X14 only.
- Jetter Ethernet system bus multicast frames of port X14 are forwarded to the CPU only.
- Jetter Ethernet system bus multicast frames of port X15 are forwarded to the CPU and to port X14.
- All other Ethernet frames are forwarded as usual.

**Purpose:**

This command lets you prevent forwarding Jetter Ethernet system bus multicast frames to port X15. This way, networks are split and thus data traffic - e.g. from the machine network to higher-level networks - is reduced.

**Address space**

Splitting is carried out on Ethernet level via the multicast address range of the Jetter Ethernet system bus.
0x01 00 5E 40 00 00 ... 0x01 00 5E 40 00 FF

# 10.4 General system registers

**Introduction**

In this chapter, the system command registers and the system commands will be explained in detail.

**Contents**


| Topic | Page |
|---|---|
| Description of system command registers | 303 |
| Description of system commands | 306 |

# Description of system command registers

**Registers - Overview**

The following registers are described in this manual:

| Register | Description |
|---|---|
| **R 202960** | System password register |
| **R 202961** | System command register |
| **R 202962** | System status register |

**R 202960**

## System password register

Enter system password 1112502132 (0x424F6F74) into this register. Then enter the required command value into the system command register. Now, the controller sets the value of this register to 0.

| **Register properties** | |
|---|---|
| Value | 1112502132 (0x424F6F74) |

**R 202961**

## System command register

Enter the system commands into this register. Then the controller executes the command. Then, it sets the value of this register to 0.

| **Commands** | |
|---|---|
| **102** | **Restart the controller** |
| **104** | **Reset remanent parameters** |
| **122** | **Deactivate - Wait for communication** |
| **123** | **Activate - Wait for communication** |
| **160** | **Deactivate - Task switch on I/O access** |
| **161** | **Activate - Task switch on I/O access** |
| **170** | **Deactivate - Resume task time slot** |

| Commands | |
|---|---|
| **171** | **Activate - Resume task time slot** |
| **310** | **Load configuration files** |
| **311** | **Load module configuration** |
| **312** | **Load process data configuration for Ethernet system bus** |
| **313** | **Stop process data communication for Ethernet system bus** |
| **330** | **Disable JetIPScan client** |
| **331** | **Enable JetIPScan client** |
| **410** | **Disable JetSync blockage** |
| **411** | **Enable JetSync blockage for all ports** |
| **412** | **Enable JetSync blockage for port X15** |

| Register properties | |
|---|---|
| Access | System password register contains the correct password. |

**R 202962**

## System status register

The system status register lets you evaluate the system conditions.

| Meaning of the individual bits | | |
|---|---|---|
| **Bit 0** | **Task switch on I/O access** | |
| | 0 = | No task switching in the application program on I/O access. |
| | 1 = | Task switching is carried out in the application program on I/O access. |
| **Bit 1** | **Wait for communication** | |
| | 0 = | The controller waits for communication requests for a short time. |
| | 1 = | The controller does not wait for communication requests. |
| **Bit 2** | **JetIPScan client** | |
| | 0 = | JetIPScan client is not active |
| | 1 = | JetIPScan client is active |
| **Bit 8** | **JetSync blockage** | |
| | 0 = | JetSync blockage is not active |
| | 1 = | JetSync blockage is active |

| Register properties | |
|---|---|
| Access | Read |

# Description of system commands

**System command 102**

**Restart the controller**

**Effect:**

The controller is restarting. The effect is the same as when you switch the power supply off and on again.

**Purpose:**

Use this command, for example, if you have made changes to system registers or system files which become active only when the controller is rebooted.

**System command 104**

**Reset remanent parameters**

**Effect:**

The controller will reset remanent parameters to their default values (factory settings).

| Register number | Description | Factory settings |
|---|---|---|
| 100002023 | JX3 system bus: I/O dummy modules | 65535 |
| 100002034 | JX3 system bus: Number of retries | 1 |
| 200002023 | JX2 system bus: I/O dummy modules | -1 |
| 200002024 | JX2 system bus: Slave dummy modules | 255 |
| 200002029 | JX2 system bus: Baud rate | 7 |
| 200002032 | JX2 system bus: Switch-on delay | 60 |
| 200002077 | JX2 system bus: Special functions | 0 |

**Application:**

Use this command, if you want to undo changes to remanent parameters.

**System command 122**

**Deactivate - Wait for communication**

**Effect:**

Not before there are definite requests, the controller will communicate with external communication partners.

**Advantage:**

The controller executes the application program faster.

**Disadvantage:**

On average, external communication partners have to wait longer for a response from the controller.

**System command 123** — **Activate - Wait for communication**

**Effect:**

The controller cyclically checks for communication requests from external partners for 1 to 2 ms.

**Advantage:**

External communication partners get a faster reply from the controller.

**Disadvantage:**

Application program processing takes slightly longer.

**System command 160** — **Deactivate - Task switch on I/O access**

**Effect:**

While the controller is accessing modules on the JX2 or JX3 system bus, other tasks of the application program are not processed.

**Advantage:**

The controller executes I/O accesses as fast as possible.

**Disadvantage:**

As certain I/O accesses are significantly slower than access to internal variables, response time of other tasks may increase.

**System command 161** — **Activate - Task switch on I/O access**

**Effect:**

While the controller is accessing modules on the JX2 or JX3 system bus, it processes the other tasks of the application program.

**Advantage:**

The execution time of certain I/O accesses which may be relatively long does not affect the response time of other tasks.

**Disadvantage:**

The run time of the other tasks influences the execution time of several I/O accesses.

**System command 170** — **Deactivate - Resume task time slot**

**Effect:**

When a normal application task has been interrupted by a cyclic task or the Ethernet system bus publisher, processing the following application task is resumed. The remaining time of the time slot of the interrupted task lapses for one cycle.

**Advantage:**

The total cycle time for processing all tasks is not influenced so much by the cyclic events.

**Disadvantage:**

This way, the interrupted task is assigned less runtime.

**System command 171** — **Activate - Resume task time slot**

**Effect:**

When a normal application task has been interrupted by a cyclic task or the Ethernet system bus publisher, processing the interrupted application task is resumed. This way, the interrupted task is processed for the remaining time of its time slot.

**Advantage:**

The interrupted task is assigned its total runtime.

**Disadvantage:**

The total cycle time for processing all tasks is influenced by the cyclic events to a greater extend.

**System command 310** — **Load configuration files**

**Effect:**

The controller loads the module configuration file (ModConfig.da) and the configuration files for process data communication on the Jetter Ethernet system bus (Publisher.pub, Subscriber.sub) from the file system. This corresponds to a combination of commands 311 and 312.

**Purpose:**

Once the transfer of these files into the controller's file system is completed, system command 310 enables their contents.

**System command 311** — **Load module configuration**

**Effect:**

The controller loads the module configuration file (ModConfig.da) from the file system.

**Purpose:**

Once the transfer of this file into the controller's file system is completed, system command 311 enables its contents.

**System command 312** — **Load process data configuration for Ethernet system bus**

**Effect:**

The controller loads the configuration files for process data communication on the Jetter Ethernet system bus (Publisher.pub, Subscriber.sub) from the file system.

**Purpose:**

Once the transfer of these files into the controller's file system is completed, system command 312 enables their contents.

**System command 313** — **Stop process data communication for Ethernet system bus**

**Effect:**

Process data communication on the Jetter Ethernet system bus stops.

**Purpose:**

Transfer the configuration files for process data communication on the Jetter Ethernet system bus into the controller's file system. Then, stop process data communication by issuing system command 313. Finally, enable the contents of the new files.

**System command 330** — **Disable JetIPScan client**

**Effect:**

This command lets you disable the JetIPScan client. The server, however, remains enabled.

**Purpose:**

For testing purposes

**System command 331** — **Enable JetIPScan client**

**Effect:**

This command lets you enable the JetIPScan client.

**Purpose:**

This command lets you enable the JetIPScan client which has been disabled for testing purposes.

**System command 410** — **Disable JetSync blockage**

**Effect:**

- The JetSync blockage is disabled for all ports. Bit 8 in R 202962 is reset.
- The Jetter Ethernet system bus multicast frames are transmitted to all ports (X14, X15 and CPU).

**Purpose:**

The JetSync blockage enabled by system command 411 or 412 is disabled. Forwarding the Jetter Ethernet system bus multicast frames to all ports again corresponds to the on-state of the controller.

**System command 411** — **Enable JetSync blockage for all ports**

**Effect:**

- The JetSync blockage is enabled for all ports (X14, X15, and CPU). Bit 8 in R 202962 is set.

- Jetter Ethernet system bus multicast frames which are received on a port are not forwarded to any of the other ports.
- All other Ethernet frames are forwarded as usual.

**Purpose:**

This command lets you prevent forwarding Jetter Ethernet system bus multicast frames to the CPU and the other ports. This way, networks are split and thus data traffic - e.g. from the machine network to higher-level networks - is reduced.

**Address space**

Splitting is carried out on Ethernet level via the multicast address range of the Jetter Ethernet system bus.
0x01 00 5E 40 00 00 ... 0x01 00 5E 40 00 FF

**System command 412** **Enable JetSync blockage for port X15**

**Effect:**

- The JetSync blockage is enabled for port X15 only. Bit 8 in R 202962 is set.
- Jetter Ethernet system bus multicast frames of the CPU are forwarded to port X14 only.
- Jetter Ethernet system bus multicast frames of port X14 are forwarded to the CPU only.
- Jetter Ethernet system bus multicast frames of port X15 are forwarded to the CPU and to port X14.
- All other Ethernet frames are forwarded as usual.

**Purpose:**

This command lets you prevent forwarding Jetter Ethernet system bus multicast frames to port X15. This way, networks are split and thus data traffic - e.g. from the machine network to higher-level networks - is reduced.

**Address space**

Splitting is carried out on Ethernet level via the multicast address range of the Jetter Ethernet system bus.
0x01 00 5E 40 00 00 ... 0x01 00 5E 40 00 FF

# 10.5 Startup delay register

**Introduction**

The device JC-360 provides a register to which a delay time can be written.

**Application**

The boot process of the device is delayed by the entered delay time.

**Contents**


| Topic | Page |
|---|---|
| Setting the startup delay | 312 |

## Setting the startup delay

**Introduction**

If other devices connected to the bus have got a longer startup time, the boot process must be delayed.

**Set delay time**

To set the delay time, proceed as follows:

| Step | Action |
|---|---|
| **1** | Switch on the device JC-360. |
| **2** | Enter the password. For this, write value 1112502132 (0x424f6f74) to R 202970. |
| **3** | Enter the desired delay time in steps of 100 ms into register 202971. |

**Result:** The next boot process will be delayed by the set startup delay time before initializing the JX2 and JX3 system bus.

**R 202970**

**Password register**

Enter 1112502132 (0x424F6F74) into this register. Then enter the desired value into the startup delay time register. Now, the controller sets the value of this register to 0.

| **Register properties** | |
|---|---|
| Value | 1112502132 (0x424F6F74) |

**R 202971**

**Startup delay time**

Write into this register the delay time in multiples of 100 milliseconds.

| **Register properties** | |
|---|---|
| Values | 0 (OFF) ... 3,000 (300 seconds) |
| Value after reset | As described above (remanent) |

**Procedure**

- The controller only executes start delay, when switch S11 is in *RUN* position.
- Start delay is terminated by leaving the *RUN* position.

**Display**

- LED **D1** flashing slowly during the first half of the start delay time (approx. 1 Hz).
- LED **D1** flashing fast during the second half of the start delay time (approx. 4 Hz).

# 10.6 Real-time clock (RTC)

**Introduction**

The JC-360 is equipped with a component which maintains time and date settings for a certain time even when it is not energized.

**Usage by OS**

The OS uses the real-time clock for the following functions:

- Storing file date and time

**Restrictions**

When using the real-time clock, the following restrictions apply:

- When the device is de-energized the power reserve is limited.
- The real-time clock has no automatic daylight savings time function.

**Contents**


| Topic | Page |
|---|---|
| Technical specifications | 314 |
| Programming | 315 |

# Technical specifications

**Technical data - Real-time clock**

| Parameter | Description |
|---|---|
| Power reserve,<br>if the controller has been running for at least 1 hour. | Minimum: 1 week<br><br>Typical: 2 weeks |
| Deviation | Maximum: 1 min per month |

**Behavior when the power reserve has elapsed**

If the controller has been deenergized for a longer period of time and the RTC power reserve has elapsed, the controller performs the following steps when re-booting:

| Step | Description |
|---|---|
| **1** | During the boot process the controller detects that the power reserve has elapsed. |
| **2** | The controller sets date and time to their default values:<br>Date: Saturday, January 01, 2000<br>Time: 0:00 a.m. |

**Factory settings**

At the end of the controller manufacturing process, the manufacturers set the real-time clock to the actual date and time. As the power reserve corresponds to the typical delivery time, the as delivered condition is undefined.

# Programming

**Programming using STX**

To program date and time it is advisable to use the functions provided by JetSym STX:

- DateTimeActual()
- DateTimeDecode()
- DateTimeEncode()
- DateTimeIsValid()
- DateTimeSet()

For more information on these functions refer to the JetSym online help.

If you make use of the above functions, the minimum time interval is one second. If you need a time interval of one second, programming must be made by using the registers described below.

**Programming using registers**

Depending on the respective application, access to the real-time clock via registers might be required. For this, there are two register sets:

- Register set 1 is for directly accessing individual real-time clock values.
- Changes to values in register set 1 are immediately transferred to the real-time clock.
- Register set 2 operates within a buffer. In the buffer, all real-time clock values are consistently read out and written.
- Not before the trigger register is written to, the value changes made in or out of register set 2 are transferred.

**Register overview**

The following registers have been assigned to the real-time clock:

**Register set 1: Direct access**

| Register | Description |
|---|---|
| **R 102910** | Milliseconds |
| **R 102911** | Seconds |
| **R 102912** | Minutes |
| **R 102913** | Hours |
| **R 102914** | Weekday (0 = Sunday) |
| **R 102915** | Day |
| **R 102916** | Month |
| **R 102917** | Year |

**Register set 2: Buffer access**

| Register | Description |
|---|---|
| **R 102920** | Milliseconds |
| **R 102921** | Seconds |
| **R 102922** | Minutes |
| **R 102923** | Hours |
| **R 102924** | Weekday (0 = Sunday) |
| **R 102925** | Day |
| **R 102926** | Month |
| **R 102927** | Year |
| **R 102928** | Read/write trigger |

**R 102910**

## Milliseconds

This register contains the millisecond of the actual time.

| **Register properties** | |
|---|---|
| Values | 0 ... 999 |
| Value after reset | 0 |

**R 102911**

## Seconds

This register contains the seconds of the actual time.

| **Register properties** | | |
|---|---|---|
| Values | 0 ... 59 | |
| Value after reset | **If ...** | **... then ...** |
| | the power reserve has not elapsed, | actual time |
| | the power reserve has elapsed, | 0 |

**R 102912**

## Minutes

This register contains the minutes of the actual time.

| Register properties | | |
|---|---|---|
| Values | 0 ... 59 | |
| Value after reset | **If ...** | **... then ...** |
| | the power reserve has not elapsed, | actual time |
| | the power reserve has elapsed, | 0 |

**R 102913**

## Hours

This register contains the hours of the actual time.

| Register properties | | |
|---|---|---|
| Values | 0 ... 23 | |
| Value after reset | **If ...** | **... then ...** |
| | the power reserve has not elapsed, | actual time |
| | the power reserve has elapsed, | 0 |

**R 102914**

## Weekday

This register contains the weekday of the actual date.

| Register properties | | |
|---|---|---|
| Values | 0 ... 6 (0 = Sunday) | |
| Value following a reset | **If ...** | **... then ...** |
| | the power reserve has not elapsed, | actual time |
| | the power reserve has elapsed, | 0 |

**R 102915**

## Day

This register contains the day of the actual date.

| Register properties | | |
|---|---|---|
| Values | 1 ... 31 | |
| Value after reset | **If ...** | **... then ...** |
| | the power reserve has not elapsed, | actual time |
| | the power reserve has elapsed, | 1 |

**R 102916**

## Month

This register contains the month of the actual date.

| Register properties | | |
|---|---|---|
| Values | 1 ... 12 | |
| Value after reset | **If ...** | **... then ...** |
| | the power reserve has not elapsed, | actual time |
| | the power reserve has elapsed, | 1 |

**R 102917**

## Year

This register contains the year of the actual date.

| Register properties | | |
|---|---|---|
| Values | 0 ... 99 | |
| Value after reset | **If ...** | **... then ...** |
| | the power reserve has not elapsed, | actual time |
| | the power reserve has elapsed, | 0 |

**R 102920** **Milliseconds**

This register contains the milliseconds stored in the buffer.

| Register properties | |
|---|---|
| Values | 0 ... 999 |
| Value after reset | 0 |
| Takes effect | After read/write access to register 102928 |

**R 102921** **Seconds**

This register contains the seconds stored in the buffer.

| Register properties | |
|---|---|
| Values | 0 ... 59 |
| Value after reset | 0 |
| Takes effect | After read/write access to register 102928 |

**R 102922** **Minutes**

This register contains the minutes stored in the buffer.

| Register properties | |
|---|---|
| Values | 0 ... 59 |
| Value after reset | 0 |
| Takes effect | After read/write access to register 102928 |

**R 102923** **Hours**

This register contains the hours stored in the buffer.

| Register properties | |
|---|---|
| Values | 0 ... 23 |
| Value after reset | 0 |
| Takes effect | After read/write access to register 102928 |

**R 102924** **Weekday**

This register contains the weekday stored in the buffer.

| Register properties | |
|---|---|
| Values | 0 ... 6 (0 = Sunday) |
| Value following a reset | 0 |
| Takes effect | After read/write access to register 102928 |

**R 102925** **Day**

This register contains the day stored in the buffer.

| Register properties | |
|---|---|
| Values | 0 ... 31 |
| Value after reset | 0 |
| Takes effect | After read/write access to register 102928 |

**R 102926** **Month**

This register contains the month stored in the buffer.

| Register properties | |
|---|---|
| Values | 0 ... 12 |
| Value after reset | 0 |
| Takes effect | After read/write access to register 102928 |

**R 102927** **Year**

This register contains the year stored in the buffer.

| Register properties | |
|---|---|
| Values | 0 ... 99 |
| Value after reset | 0 |
| Takes effect | After read/write access to register 102928 |

**R 102928** **Read/write trigger**

This register allows transferring values between buffer register and real-time clock.

| Register properties | |
|---|---|
| Read | The actual date and time are transferred from real-time clock to buffer registers 102920 through 102927.<br>The reading is undefined. |
| Write | The values contained in buffer registers 102920 ... 102927 are transferred to the real-time clock.<br>The value written is ignored. |

# 10.7 Runtime registers

**Introduction**

The JC-360 provides several registers which are incremented by the operating system at regular intervals.

**Application**

These registers can be used to easily carry out time measurements in the application program.

**Contents**


| Topic | Page |
|---|---|
| Description of the runtime registers | 323 |

# Description of the runtime registers

**Register overview**

The device is equipped with the following runtime registers:

| Register | Description |
|---|---|
| **R 201000** | Application time base in milliseconds |
| **R 201001** | Application time base in seconds |
| **R 201002** | Application time base in R 201003 * 10 ms |
| **R 201003** | Application time base units for R 201002 |
| **R 201004** | System time base in milliseconds |
| **R 201005** | System time base in microseconds |

**R 201000**

## Application time base in milliseconds

Every millisecond this register is incremented by one.

| Register properties | |
|---|---|
| Values | -2,147,483,648 ... 2,147,483,647 (overflowing) |

**R 201001**

## Application time base in seconds

Every second this register is incremented by one.

| Register properties | |
|---|---|
| Values | -2,147,483,648 ... 2,147,483,647 (overflowing) |

**R 201002**

## Application time base in application time base units

Every [R 201003] * 10 ms this register value is incremented by one. Using the reset value 10 in register 201003, this register is incremented every 100 ms.

| Register properties | |
|---|---|
| Values | -2,147,483,648 ... 2,147,483,647 (overflowing) |

**R 201003** **Application time base units for R 201002**

This register contains the multiplier for runtime register R 201002.

| Register properties | |
|---|---|
| Values | 1 ... 2,147,483,647 (* 10 ms) |
| Value after reset | 10 (--> 100 ms) |
| Enabling conditions | After at least 10 ms |

**R 201004** **System time base in milliseconds**

Every millisecond this register value is incremented by one.

| Register properties | |
|---|---|
| Values | -2,147,483,648 ... 2,147,483,647 (overflowing) |
| Type of access | Read |

**R 201005** **System time base in microseconds**

Every microsecond this register value is incremented by one.

| Register properties | |
|---|---|
| Values | -2,147,483,648 ... 2,147,483,647 (overflowing) |
| Type of access | Read |

# 10.8 Monitoring interface activities

**Introduction**

Several servers for variables have been integrated into the controller to make variables used within the controller accessible from outside. These servers support several protocols on different interfaces. The servers do not require any programming in the application program, but process requests from external clients on their own.

This chapter explains one possibility for detecting from within the application program whether communication with the servers takes place through these interfaces.

**Monitored interface activities**

The following interface activities can be monitored:

- pcomX server via serial interface
- JetIP server via Ethernet interface
- STX debug server via Ethernet interface

**Purpose**

The monitoring function for interface activities can be used, amongst others, for the following scenarios:

- Plants requiring process visualization to ensure safe operation. They can be transferred into a save state if communication fails.
- When the service technician connects an HMI, the application program automatically displays additional status information.

**Contents**


| Topic | Page |
|---|---|
| Operating principle | 326 |
| Programming | 328 |

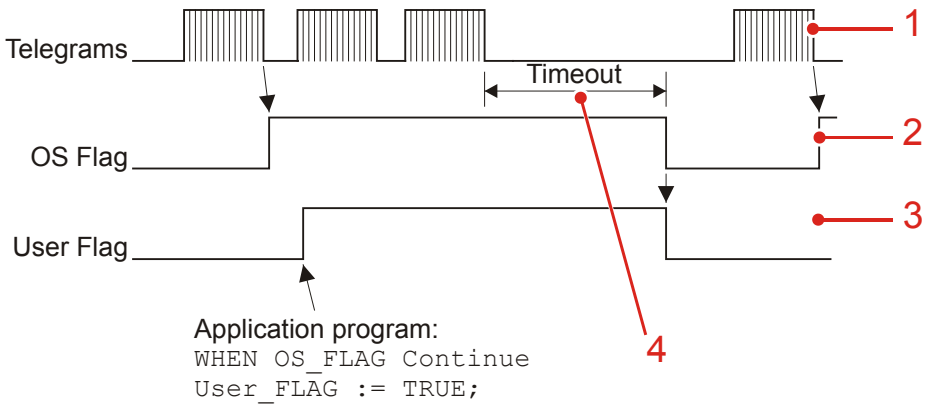
# Operating principle

**Introduction**

The application program monitors the activity of a client communicating with a server of the device JC-360 by means of two special flags and one special register per interface.

**Overview**

The illustration below shows the interdependence between interface activity and the two special flags, as well as the special register:




| Number | Element | Description |
|---|---|---|
| **1** | Telegrams | The client places requests to the server. |
| **2** | OS flag | OS flag set by the device JC-360 after receiving a request |
| **3** | User flag | You must set the user flag in the application program once the device has set the OS flag.This indicates that the connection has temporarily been disrupted even if the device resets the OS flag very quickly. |
| **4** | Timeout | Time of inactivity after which the OS resets both special flags. This time can be set in a special register. |

**Description**

Interface activities are monitored as follows:

| Step | Description |
|---|---|
| **1** | Enter the desired value into the timeout register of the application program. This way, the monitoring mode is activated as well. |
| **2** | When the controller receives the next telegram, the device JC-360 sets the corresponding OS flag. |
| **3** | If the OS flag has been set, the application program also sets the respective user flag. |
| **4** | Each new telegram causes the timeout to restart. |

| Step | Description |
|---|---|
| **5** | If telegrams cease to arrive, both special flags are reset by the controller upon expiry of the timeout interval. |
| **6** | The application program detects that the device has reset the special flags and therefore takes appropriate action. |
| **7** | When further telegrams start arriving, the device sets the corresponding OS flag. The user flag, however, remains reset. |

# Programming

**Registers/flags - Overview**

For interface monitoring, the device provides the following registers and flags:

**Timeout registers**

| Register | Interface | Use |
|---|---|---|
| **R 203000** | JetIP (Ethernet) | ▪ Visualization<br>▪ Controller networking |
| **R 203001** | pcomX (serial interface) | ▪ HMIs with alphanumeric display<br>▪ JetSym via serial interface |
| **R 203005** | STX debugging (Ethernet) | ▪ JetSym via Ethernet |

**Special flags**

| Flags | Interface | Use |
|---|---|---|
| **F 2088** | JetIP (Ethernet) | OS flag |
| **F 2089** | | User flag |
| **F 2090** | pcomX (serial interface) | OS flag |
| **F 2091** | | User flag |
| **F 2098** | STX debugging (Ethernet) | OS flag |
| **F 2099** | | User flag |

**R 203000**

**Timeout in the case of JetIP (Ethernet)**

This register contains the timeout for the JetIP server (Ethernet) in milliseconds.

| **Register properties** | |
|---|---|
| Values | 0 ... 2,147,483,647 [ms] |
| Value after reset | 0 (monitoring disabled) |

**R 203001**

**Timeout in the case of pcomX (serial interface)**

This register contains the timeout period for the pcomX server (serial interface) in milliseconds.

| **Register properties** | |
|---|---|
| Values | 0 ... 2,147,483,647 [ms] |
| Value after reset | 0 (monitoring disabled) |

**R 203005** **Timeout in the case of STX debugging (Ethernet)**

This register contains the timeout for the STX debug server (Ethernet) in milliseconds.

| Register properties | |
|---|---|
| Values | 0 ... 2,147,483,647 [ms] |
| Value after reset | 0 (monitoring disabled) |

**Enabling the monitoring function**

To enable monitoring of interface activities, proceed as follows:

| Step | Action |
|---|---|
| **1** | Enter the desired value into the timeout register of this interface. |
| **2** | Wait until the controller has set the OS flag of this interface. |
| **3** | Set the corresponding user flag. |

**Detecting a timeout**

To detect a timeout, proceed as follows:

| Step | Action | |
|---|---|---|
| **1** | Enable monitoring of interface activities (see above). | |
| **2** | Wait until the controller has reset the user flag of this interface.<br>**Result**: A timeout has occurred. | |
| **3** | Check the corresponding OS flag. | |
| | **If ...** | **... then ...** |
| | ... the OS flag is set, | ... the connection was temporarily disrupted. |
| | ... the OS flag is reset, | ... the connection is still disrupted. |

# 10.9 Controlling HMIs with alphanumeric displays

**Introduction**

This chapter contains information on how to control HMIs with text displays using the application program in a JC-360 controller. It also describes the registers used to parameterize the display functions. The controller provides the following display functions:

- Displaying texts
- Displaying the contents of variables
- Scanning the HMI keys
- Switching the HMI LEDs
- Monitor function

**Prerequisites**

In this manual we proceed from the assumption that the user is familiar with the following STX instructions `DisplayText()`, `DisplayText2()`, `DisplayValue()`, and `UserInput()`. For a more detailed description of the instructions mentioned above refer to the online help included in the programming tool JetSym.

**Contents**


| Topic | Page |
|---|---|
| Connectable HMIs | 331 |
| Registers | 333 |
| Configuring the screen size | 337 |
| Displaying texts | 339 |
| Displaying numerical values | 348 |
| Entering numerical values | 358 |
| Querying the keys | 371 |
| Activating/deactivating LEDs | 378 |
| Monitor functions | 382 |

## 10.9.1 Connectable HMIs

**Introduction**

This chapter lists the HMIs by Jetter AG which you can connect to the JC-360.

**Connection**

For a detailed description on how to connect these HMIs to the controller refer to chapter Mounting and installation, *Connecting HMIs* (see page 130)

**Contents**


| Topic | Page |
|---|---|
| Overview of displays and HMIs | 332 |

# Overview of displays and HMIs

**List of displays and HMIs**

The following table lists the alphanumeric displays HMIs by Jetter AG which you can connect to the JC-360.

| Order reference | Display | Keys | Interface cable |
|---|---|---|---|
| **LCD 16** | 4 lines of 20 characters each | ■ 5 function keys with LED<br>■ Can be expanded by a keyboard module NUM25 | JC-DK-Xm |
| **LCD 23** | 2 lines of 24 characters each | ■ Cursor left<br>■ Cursor right<br>■ ENTER ([↵]) | JC-DK-Xm |
| **LCD 27** | 2 lines of 24 characters each | ■ 5 function keys<br>■ Cursor keypad<br>■ Clear<br>■ ENTER ([↵]) | JC-DK-Xm |
| **LCD 34** | 2 lines of 24 characters each | ■ 5 function keys<br>■ Numeric keypad | JC-DK-Xm |
| **LCD 52** | 4 lines of 16 characters each | ■ 6 function keys<br>■ Numeric keypad | KAY-0533-0025 |
| **LCD 54** | 4 lines of 16 characters each | ■ 8 function keys<br>■ Numeric keypad<br>■ Emergency stop | KAY-0533-0025 |
| **LCD 54Z** | 4 lines of 16 characters each | ■ 8 function keys<br>■ Numeric keypad<br>■ Emergency stop<br>■ Two-hand control device | KAY-0533-0025 |
| **LCD 60** | 2 lines of 40 characters each | ■ 8 function keys with LED<br>■ Numeric keypad | KAY-0386-xxxx |
| **LCD 110** | 4 lines of 20 characters each | ■ 8 function keys with LED<br>■ Numeric keypad | JC-DK-Xm |

## 10.9.2 Registers

**Introduction**

This chapter gives an overview of registers by which you can parameterize the control of HMIs or query status information.

**Restrictions**

The settings made in this register are of global effect, that is, they will impact all functions for controlling HMIs. If different settings are used in several tasks of the application program, then these settings may impact each other.

**Contents**


| Topic | Page |
|---|---|
| Register numbers | 334 |
| Registers - Overview | 335 |

# Register numbers

**Introduction**

The registers are combined into one register block. The basic register number of this block is dependent on the controller.

**Register numbers**

| Device | Basic register number | Register numbers |
|---|---|---|
| JC-360 | 220000 | 222804 ... 222840 |

**Determining register numbers**

In this chapter, only the last four figures of a register number are specified. e.g. MR 2815. Add to this module register number the basic register number of the corresponding device to determine the complete register number, for example 222815.

# Registers - Overview

**Registers - Overview**

The following table provides an overview of existing registers. For a detailed description of these registers refer to the following chapters.

| Registers | Description |
|---|---|
| **MR 2804** | Number of characters on the screen |
| **MR 2805** | Number of characters per line |
| **MR 2806** | Text selection (`DisplayText2()`) |
| **MR 2808** | Number of decimal places (`UserInput()`) |
| **MR 2810** | Number of decimal places (`DisplayValue()`) |
| **MR 2811** | Maximum number of decimal places (`UserInput()`) |
| **MR 2812** | Field length (`DisplayValue()`) |
| **MR 2813** | Field length (`UserInput()`) |
| **MR 2814** | Indirect cursor position |
| **MR 2815** | Suggested value (`UserInput()`) |
| **MR 2816** | Displaying the sign |
| **MR 2817** | State of the UserInput (`UserInput()`) |
| **MR 2818** | Monitor functions (enable/disable) |
| **MR 2819** | Display time for monitor functions |
| **MR 2820** | Switch-over to monitor display |
| **MR 2821** | Dialog language of the monitor function |
| **MR 2824** | Indirect device number (default device) |
| **MR 2825** | Device number for HMI 1 (multi-display mode) |
| **MR 2826** | Device number for HMI 2 (multi-display mode) |
| **MR 2827** | Device number for HMI 3 (multi-display mode) |
| **MR 2828** | Device number for HMI 4 (multi-display mode) |
| **MR 2829** | Basic flag number for HMI 1 (multi-display mode) |
| **MR 2830** | Basic flag number for HMI 2 (multi-display mode) |
| **MR 2831** | Basic flag number for HMI 3 (multi-display mode) |
| **MR 2832** | Basic flag number for HMI 4 (multi-display mode) |
| **MR 2833** | Register number for LEDs on HMI 1 (multi-display mode) |
| **MR 2834** | Register number for LEDs on HMI 2 (multi-display mode) |

| Registers | Description |
|---|---|
| **MR 2835** | Register number for LEDs on HMI 3 (multi-display mode) |
| **MR 2836** | Register number for LEDs on HMI 4 (multi-display mode) |
| **MR 2837** | Module number - Printer module |
| **MR 2838** | Module number - Serial interface module |
| **MR 2839** | Control character for deleting the screen |
| **MR 2840** | Control character for deleting the screen up to the end of a line |

## 10.9.3 Configuring the screen size

**Introduction**

This chapter gives a description on how the screen size of the HMI is configured in the controller.

**Why do I have to configure the screen size?**

During the boot process the HMI logs in to the controller and transmits its display size. This way, the controller is able to configure the size automatically. Therefore, you do not have to configure the display size manually. But in some cases this feature might make sense.

**Why is the correct screen size important?**

To ensure that the controller correctly executes the special functions *Delete Screen*, and *Delete to End of Line* when displaying texts (DisplayText instructions).

**Contents**


| Topic | Page |
|---|---|
| Configuring the screen size manually | 338 |

# Configuring the screen size manually

**Configuring the screen size manually**

To configure the screen size manually, proceed as follows:

| Step | Action |
|---|---|
| **1** | Enter the number of characters per line into MR 2805. |
| **2** | Multiply the value contained in MR 2805 by the number of lines and enter the result into MR 2804. |

**MR 2804**

**Number of characters on the screen**

This module register contains the number of characters displayed on the screen.

| **Module register properties** | |
|---|---|
| Values | 1 ... 128 |
| Value after reset | 48 |

**MR 2805**

**Number of characters per line**

This module register contains the number of characters per line.

| **Module register properties** | |
|---|---|
| Values | 1 ... 128 |
| Value after reset | 24 |

# 10.9.4 Displaying texts

**Introduction**

This chapter describes displaying texts on HMIs and how to parameterize the corresponding STX instructions.

**STX instructions**

In order to display texts, use the following STX instructions (STX functions):

- `DisplayText()`
- `DisplayText2()`

**Contents**


| Topic | Page |
|---|---|
| STX Instructions for displaying texts | 340 |
| Device numbers | 342 |
| Cursor position | 344 |
| Clearing the screen | 346 |

## STX Instructions for displaying texts

**Function declaration**

```
Function DisplayText(Dev:Int,
                     Pos:Int,
                     Const Ref Text:String);
```

**Function parameters**

| Parameter | Value | Description |
|---|---|---|
| Dev | 0 ... 4 | Number of the device on which the text is to be output |
| Pos | 1 ... Possible number of characters on the screen | Cursor position starting from which the text is to be displayed. |
| Text | Text to be displayed | Constant text, or name of a string variable |

**How to use this instruction**

How to invoke the instruction to display a text:

```
DisplayText(0, 1, '_Hello World!');
DisplayText(0, 25, StringVar);
```

**How it works**

The first STX instruction deletes the entire content of the screen ('_' in text). Then, this instruction causes the text 'Hello World!' to be displayed starting at cursor position1. The second STX instruction causes the content of the string variable **StringVar** to be displayed starting at cursor position 25. Both texts are displayed on the default device (Dev = 0).

**Function declaration**

```
Function DisplayText2(Dev:Int,
             Pos:Int,
             Const Ref Text1:String,
             Const Ref Text2:String);
```

**Function parameters**

| Parameter | Value | Description |
|---|---|---|
| Dev | 0 ... 4 | Number of the device on which the text is to be output |
| Pos | 1 ... Number of characters on the screen | Cursor position starting from which the text is to be displayed. |
| Text1 | Text to be displayed | Constant text, or name of a string variable |
| Text2 | Text to be displayed | Constant text, or name of a string variable |

**How to use this instruction**

How to invoke the instruction to display one of two texts:

```
DisplayText2(0, 25, 'Fehler:', 'Error:');
```

**How it works**

On the default device (Dev = 0) starting from cursor position 25, the STX instruction causes the text 'Fehler:' or the text 'Error:' to be displayed. MR 2806 lets you control which text will be displayed.

**MR 2806**

**Text selection for DisplayText2**

The value in this module register specifies which one of the two texts is to be displayed.

| Module register properties | | |
|---|---|---|
| Values | 0 | Text1 |
| | 1 | Text2 |

# Device numbers

**Introduction**

The device number lets you define the HMI.

**Device numbers**

You may enter the following values for the parameter device number:

| Number | Device | Description |
|---|---|---|
| **0** | Default device | MR 2824 holds the number of the device to be used. |
| **1** | HMI # 1 | Multi-display mode |
| **2** | HMI # 2 | Single-/Multi-display mode |
| **3** | HMI # 3 | Multi-display mode |
| **4** | HMI # 4 | Multi-display mode |
| **5 ... 7** | Reserved | Do not use |
| **8** | Printer module | Output of data on a printer module connected to the JX2 or JX3 system bus |
| **9, 10** | Serial interface | Output of data on a user-programmable serial interface |
| **11** | Serial interface module | Output of data on a serial interface module connected to the JX2 or JX3 system bus |

**MR 2824**

**Device number of the default device**

This module register contains the device number of the default device. If you always specify the default device in the application program (device number = 0), you can select during runtime which device is actually to be used.

| **Module register properties** | |
|---|---|
| Values | 1 ... 11 |
| Value after reset | 2 |

**Single-display mode**

In single-display mode, an HMI always displays data from device no. **2**.

**Multi-display mode**

In multi-display mode, an HMI always displays data from the device the number of which is contained in the corresponding configuration register MR 2825 through MR 2828.

**MR 2825** **Device number for HMI 1 in multi-display mode.**

| Module register properties | |
|---|---|
| Values | 1 ... 4 |
| Value after reset | 1 |

**MR 2826** **Device number for HMI 2 in multi-display mode.**

| Module register properties | |
|---|---|
| Values | 1 ... 4 |
| Value after reset | 2 |

**MR 2827** **Device number for HMI 3 in multi-display mode.**

| Module register properties | |
|---|---|
| Values | 1 ... 4 |
| Value after reset | 3 |

**MR 2828** **Device number for HMI 4 in multi-display mode.**

| Module register properties | |
|---|---|
| Values | 1 ... 4 |
| Value after reset | 4 |

# Cursor position

**Introduction**

This parameter is used to define the screen position where the first character of the text or variable is to appear.

**Restrictions**

There is no cursor position evaluation for devices 8 through 11.

**Cursor position 0**

Cursor position **0** has a special meaning. The controller evaluates this parameter in the following steps:

| Step | Description | |
|---|---|---|
| 1 | The controller reads the content of MR 2814. | |
| 2 | **If ...** | **... then ...** |
| | ... MR 2814 is larger than 0, | ... the value is used as cursor position. |
| | ... MR 2814 equals 0, | ... the message is displayed starting from the current cursor position. |

**Example: MR 2814 = 0**

```
DisplayText(0, 1, 'Temp :');
DisplayValue(0, 0, Temperature);
```

**Result**: The temperature is displayed directly after the colon starting from cursor position 7.

**MR 2814**

**Indirect cursor position**

This module register specifies the cursor position, if **0** has been programmed as parameter.

| **Module register properties** | |
|---|---|
| Values | 0 ... Number of characters on the screen |
| Value after reset | 0 |

**Cursor position of HMIs**

The following table shows the correlation between cursor position as parameter of an instruction and the position on the screen.

| Designation | Line | Cursor position |
|---|---|---|
| LCD 16, LCD 110 | 1 | 1 ... 20 |
| | 2 | 21 ... 40 |
| | 3 | 41 ... 60 |
| | 4 | 61 ... 80 |
| LCD 23, LCD 27 | 1 | 1 ... 24 |
| LCD 34 | 2 | 25 ... 48 |
| LCD 52, LCD 54(Z) | 1 | 1 ... 16 |
| | 2 | 17 ... 32 |
| | 3 | 33 ... 48 |
| | 4 | 49 ... 64 |
| LCD 60 | 1 | 1 ... 40 |
| | 2 | 41 ... 80 |

## Clearing the screen

**Introduction**

In the case of text displays there are two control characters allowing to clear the screen:

- Totally clearing the screen
- Clearing the screen to the end of the line

**Restrictions**

If these display instructions are used for devices from 8 through 11, theses characters are not considered as control characters but are displayed as text.

**Clearing the screen**

The default character for deleting the whole screen is the underline character "_".
When this character is used, first, the displayed text is deleted. Then, the given text is displayed starting from cursor position 1.

**Example:**

```
DisplayText(0, 10, 'H_ello');
```

**Result**: The screen is cleared and the word fragment "ello" is displayed starting from cursor position 1.

**Delete text to the end of the line**

The default character for deleting text up to the end of line is the dollar symbol "$".

This character causes the rest of the line to be cleared, starting from the present cursor position.

**Example:**

```
DisplayText(0, 25, 'Position :$');
```

**Result:** Starting from cursor position 25, "Position :" is displayed, and the rest of the line is cleared.

**Changing control characters**

If underline and dollar symbol are to be displayed as characters, you have to change the corresponding control character. Define the control characters in module registers MR 2839, and MR 2840.

**MR 2839**

## Control character for clearing the screen

This module register contains the ASCII code of the control character which causes the screen to be cleared.

| Module register properties | |
|---|---|
| Values | 0 ... 255 |
| Value after reset | 95 ('_') |
| Takes effect | Next time when STX instruction `DisplayText()` or `DisplayText2()` is issued |

**MR 2840**

## Control character for clearing the screen up to the end of a line

This module register contains the ASCII code of the control character which causes the screen to be cleared up to the end of a line.

| Module register properties | |
|---|---|
| Values | 0 ... 255 |
| Value after reset | 36 ('$') |
| Takes effect | Next time when STX instruction `DisplayText()` or `DisplayText2()` is issued |

# 10.9.5 Displaying numerical values

**Introduction**

This chapter describes displaying numerical values on HMIs and how to parameterize the corresponding STX instruction. These numerical values may be constants or contents of registers and variables.

**STX instruction**

To display numerical values, use the following STX instruction:

- `DisplayValue()`

**Formatting the display format**

The display format of numerical values can be adapted to suit your needs. The following parameters can be adjusted:

- Display length
- Number of decimal places
- With or without sign
- Displaying values in decimal or hexadecimal notation

**Displaying numerical values**

When displaying numerical values the following formatting applies:

- The numerical value is displayed right-aligned.
- The sign is the first character which is output if the sign has not been disabled before.
- Positive values are preceded by a space character as sign. Negative values are preceded by "-".
- If the display field is too small, the leftmost figures are truncated.
- The value is rounded to the set decimal places.

**Contents**


| Topic | Page |
|---|---|
| STX instruction for displaying numerical values | 349 |
| Device numbers | 350 |
| Cursor position | 352 |
| Setting the length of the display field | 354 |
| Setting the sign option | 355 |
| Setting the number of decimal places | 356 |
| Setting the format of numerical values | 357 |

# STX instruction for displaying numerical values

**Function declaration**

```
Function DisplayValue(Dev:Int,
                      Pos:Int,
                      Value:Double);
```

**Function parameters**

| Parameter | Value | Description |
|---|---|---|
| Dev | 0 ... 4 | Number of the device where the value is to be output |
| Pos | 1 ... Possible number of characters on the screen | Cursor position starting from which the value is to be displayed. |
| Value | Value to be displayed | Constant value, name of a register or a variable |

**How to use this instruction**

How to invoke the instruction to display a value:

```
DisplayValue(0, 1, -12.345);
DisplayValue(0, 25, Axis2.Position);
```

**How it works**

The first STX instruction causes the value **-12,345** to be displayed starting at cursor position 1. The second STX instruction causes the content of the variable **Axis2.Position** to be displayed starting at cursor position 25.

Both numerical are displayed on the default device (Dev = 0).

# Device numbers

**Introduction**

The device number lets you define the HMI.

**Device numbers**

You may enter the following values for the parameter device number:

| Number | Device | Description |
|---|---|---|
| **0** | Default device | MR 2824 holds the number of the device to be used. |
| **1** | HMI # 1 | Multi-display mode |
| **2** | HMI # 2 | Single-/Multi-display mode |
| **3** | HMI # 3 | Multi-display mode |
| **4** | HMI # 4 | Multi-display mode |
| **5 ... 7** | Reserved | Do not use |
| **8** | Printer module | Output of data on a printer module connected to the JX2 or JX3 system bus |
| **9, 10** | Serial interface | Output of data on a user-programmable serial interface |
| **11** | Serial interface module | Output of data on a serial interface module connected to the JX2 or JX3 system bus |

**MR 2824**

**Device number of the default device**

This module register contains the device number of the default device. If you always specify the default device in the application program (device number = 0), you can select during runtime which device is actually to be used.

| **Module register properties** | |
|---|---|
| Values | 1 ... 11 |
| Value after reset | 2 |

**Single-display mode**

In single-display mode, an HMI always displays data from device no. **2**.

**Multi-display mode**

In multi-display mode, an HMI always displays data from the device the number of which is contained in the corresponding configuration register MR 2825 through MR 2828.

**MR 2825** **Device number for HMI 1 in multi-display mode.**

| Module register properties | |
|---|---|
| Values | 1 ... 4 |
| Value after reset | 1 |

**MR 2826** **Device number for HMI 2 in multi-display mode.**

| Module register properties | |
|---|---|
| Values | 1 ... 4 |
| Value after reset | 2 |

**MR 2827** **Device number for HMI 3 in multi-display mode.**

| Module register properties | |
|---|---|
| Values | 1 ... 4 |
| Value after reset | 3 |

**MR 2828** **Device number for HMI 4 in multi-display mode.**

| Module register properties | |
|---|---|
| Values | 1 ... 4 |
| Value after reset | 4 |

# Cursor position

**Introduction**

This parameter is used to define the screen position where the first character of the text or variable is to appear.

**Restrictions**

There is no cursor position evaluation for devices 8 through 11.

**Cursor position 0**

Cursor position **0** has a special meaning. The controller evaluates this parameter in the following steps:

| Step | Description | |
|---|---|---|
| **1** | The controller reads the content of MR 2814. | |
| **2** | **If ...** | **... then ...** |
| | ... MR 2814 is larger than 0, | ... the value is used as cursor position. |
| | ... MR 2814 equals 0, | ... the message is displayed starting from the current cursor position. |

**Example: MR 2814 = 0**

```
DisplayText(0, 1, 'Temp :');
DisplayValue(0, 0, Temperature);
```

**Result**: The temperature is displayed directly after the colon starting from cursor position 7.

**MR 2814**

**Indirect cursor position**

This module register specifies the cursor position, if **0** has been programmed as parameter.

| **Module register properties** | |
|---|---|
| Values | 0 ... Number of characters on the screen |
| Value after reset | 0 |

**Cursor position of HMIs**

The following table shows the correlation between cursor position as parameter of an instruction and the position on the screen.

| Designation | Line | Cursor position |
|---|---|---|
| LCD 16, LCD 110 | 1 | 1 ... 20 |
| | 2 | 21 ... 40 |
| | 3 | 41 ... 60 |
| | 4 | 61 ... 80 |
| LCD 23, LCD 27 | 1 | 1 ... 24 |
| LCD 34 | 2 | 25 ... 48 |
| LCD 52, LCD 54(Z) | 1 | 1 ... 16 |
| | 2 | 17 ... 32 |
| | 3 | 33 ... 48 |
| | 4 | 49 ... 64 |
| LCD 60 | 1 | 1 ... 40 |
| | 2 | 41 ... 80 |

# Setting the length of the display field

**Setting the length**

MR 2812 lets you set the length of the display field for numerical values.

```
MR 2812 = Number of figures + sign [+ decimal point]
```

**Example:**

| | |
|---|---|
| Number of figures: | 6 |
| Sign (MR 2816): | 0 (yes) |
| Decimal point: | None |
| Field length (MR 2812): | 7 |
| Display: | 7 characters |

**MR 2812**

**Field length for DisplayValue**

This module register holds the length of the display field.

| **Module register properties** | |
|---|---|
| Values | 1 ... 12 |
| Value after reset | 11 |
| Takes effect | Next time when STX instruction `DisplayValue()` is issued |

# Setting the sign option

**Setting the sign option**

MR 2816 lets you set whether the sign is displayed or not.

In setting the display field length in MR 2812 continue to add the sign's place, even if no sign is to be displayed.

**Example:**

| | |
|---|---|
| Number of figures: | 6 |
| Sign (MR 2816): | 1 (no) |
| Decimal point: | None |
| Field length (MR 2812): | 7 |
| Display: | 6 characters |

**MR 2816**

**Displaying the sign**

| **Module register properties** | | |
|---|---|---|
| Values | 0 | Sign will be displayed. |
| | 1 | Sign will not be displayed. |
| Takes effect | Next time when STX instruction `DisplayValue()` is issued | |

# Setting the number of decimal places

**Setting the number of decimal places**

The number of decimal places is set in MR 2810.

If you add decimal places, you might have to adjust the length of the display field in MR 2812.

Reason: The decimal point requires one place in the display field.

**MR 2810**

**Number of decimal places for DisplayValue instructions**

This module register holds the number of decimal places when displaying numerical values.

| **Module register properties** | |
|---|---|
| Values | 0 ... 4 |
| Takes effect | Next time when STX instruction `DisplayValue()` is issued |

# Setting the format of numerical values

**Setting the format of numerical values**

In setting the format of numerical values you have the choice between decimal or hexadecimal format. Flag 2060 lets you set the format of numerical values.

**Flag 2060**

**Format of numerical values**

| **Flag properties** | | |
|---|---|---|
| Values | 0 | Decimal |
| | 1 | Hexadecimal |
| Takes effect | Next time when STX instruction `DisplayValue()` is issued | |

## 10.9.6 Entering numerical values

**Introduction**

This chapter describes input of values on HMIs, how to assign them as register and variable values and how to parameterize the corresponding STX instruction.

**STX instruction**

To input register and variable contents via HMI, use the following STX instruction:

- `UserInput()`

**Formatting the Values to be Entered**

The format of numerical values to be entered can be adapted to suit your needs. The following parameters can be adjusted:

- Length of input field
- Maximum number of decimal places
- Displaying a suggested value

**Keys to be used for entering numerical values**

| Key | Description |
|---|---|
| **[0] ... [9]** | Entering a numerical value |
| **[.] or [,]** | Entering a decimal point or decimal comma |
| **[-]** | Entering a negative/positive numerical value<br>You can press the key any time during input. |
| **[C]** | Clearing previous entries<br>Displaying a suggested value a second time |
| **ENTER ([↵])** | Terminating the input process; accepting the value |

**Restrictions**

The following restrictions apply to the STX instruction `UserInput()`:

- While inputting numerical values, the keys used for this are not mapped to the key flags.
- While the monitor function is active, the STX instruction `UserInput()` is not executed by the controller until the monitor function is completed.

**Displaying inputs**

During input, the following values are being displayed:

| Step | Description | |
|---|---|---|
| 1 | The controller shows the suggested value. The formatting parameters are used the same way as for displaying numerical values. | |
| 2 | **If ...** | **... then ...** |
| | ... you press the key**ENTER**, | ... the controller takes over the suggested value and completes executing the instruction. |
| | ... you press another key that has been used during input, | ... the controller clears the suggested value and has the value resulting from the pressed key displayed. |
| 3 | The input numerical value is displayed left-aligned in the input field, until executing the instruction is completed or aborted. | |

**Result**: After finishing the input, the latest display remains on the screen.

**Contents**


| Topic | Page |
|---|---|
| STX instruction for the input of numerical values | 360 |
| Device numbers | 361 |
| Cursor position | 363 |
| Setting the length of the input field | 365 |
| Setting the maximum number of decimal places | 366 |
| Setting the suggested value | 367 |
| Polling the number of decimal places | 368 |
| UserInput - Polling the status | 369 |
| UserInput - Aborting the instruction | 370 |

# STX instruction for the input of numerical values

**Function declaration**

```
Function UserInput(Dev:Int,
                   Pos:Int):Double;
```

**Function parameters**

| Parameter | Value | Description |
|---|---|---|
| Dev | 0 ... 4 | Number of the device where the value is to be input |
| Pos | 1 ... Possible number of characters on the screen | Cursor position starting from which the input field is to be displayed |

**Result of the function**

| Result of the function | |
|---|---|
| Type | Double |
| Value | Value that has been input |

**How to use this instruction**

You can invoke the instruction and assign its return value to a variable in the following way:

```
AutoSet[Index].Destination := UserInput(0, 10);
```

**Operating principle**

The controller processes this instruction in the following steps:

| Step | Description | |
|---|---|---|
| **1** | The controller displays the suggested value on device **0** starting from cursor position **10**, prepares an input field and activates the cursor. | |
| **2** | The task stops at the STX instruction `UserInput()`, until it is aborted or until it is completed by the user by pressing the key **ENTER** ([↵]). | |
| **3** | **If ...** | **... then ...** |
| | ... you press the key **ENTER** ([↵]), | ... the controller assigns the value that has been input to the variable and continues the task with the next instruction. |
| | ... the STX instruction `UserInput()` instruction is aborted, | ... an exception is thrown and step 4 is carried out. |
| **4** | **If ...** | **... then ...** |
| | ... an exception handling has been programmed, | ... the task proceeds with the exception handling. |
| | ... no exception handling has been programmed, | ... the task is aborted and the error is displayed in the error register. |

# Device numbers

**Introduction**

The device number lets you define the HMI.

**Device numbers**

You may enter the following values for the parameter device number:

| Number | Device | Description |
|---|---|---|
| **0** | Default device | MR 2824 holds the number of the device to be used. |
| **1** | HMI # 1 | Multi-display mode |
| **2** | HMI # 2 | Single-/Multi-display mode |
| **3** | HMI # 3 | Multi-display mode |
| **4** | HMI # 4 | Multi-display mode |
| **5 ... 7** | Reserved | Do not use |
| **8** | Printer module | Output of data on a printer module connected to the JX2 or JX3 system bus |
| **9, 10** | Serial interface | Output of data on a user-programmable serial interface |
| **11** | Serial interface module | Output of data on a serial interface module connected to the JX2 or JX3 system bus |

**MR 2824**

**Device number of the default device**

This module register contains the device number of the default device. If you always specify the default device in the application program (device number = 0), you can select during runtime which device is actually to be used.

| **Module register properties** | |
|---|---|
| Values | 1 ... 11 |
| Value after reset | 2 |

**Single-display mode**

In single-display mode, an HMI always displays data from device no. **2**.

**Multi-display mode**

In multi-display mode, an HMI always displays data from the device the number of which is contained in the corresponding configuration register MR 2825 through MR 2828.

**MR 2825** **Device number for HMI 1 in multi-display mode.**

| Module register properties | |
|---|---|
| Values | 1 ... 4 |
| Value after reset | 1 |

**MR 2826** **Device number for HMI 2 in multi-display mode.**

| Module register properties | |
|---|---|
| Values | 1 ... 4 |
| Value after reset | 2 |

**MR 2827** **Device number for HMI 3 in multi-display mode.**

| Module register properties | |
|---|---|
| Values | 1 ... 4 |
| Value after reset | 3 |

**MR 2828** **Device number for HMI 4 in multi-display mode.**

| Module register properties | |
|---|---|
| Values | 1 ... 4 |
| Value after reset | 4 |

# Cursor position

**Introduction**

This parameter is used to define the screen position where the first character of the text or variable is to appear.

**Restrictions**

There is no cursor position evaluation for devices 8 through 11.

**Cursor position 0**

Cursor position **0** has a special meaning. The controller evaluates this parameter in the following steps:

| Step | Description | |
|---|---|---|
| **1** | The controller reads the content of MR 2814. | |
| **2** | **If ...** | **... then ...** |
| | ... MR 2814 is larger than 0, | ... the value is used as cursor position. |
| | ... MR 2814 equals 0, | ... the message is displayed starting from the current cursor position. |

**Example: MR 2814 = 0**

```
DisplayText(0, 1, 'Temp :');
DisplayValue(0, 0, Temperature);
```

**Result**: The temperature is displayed directly after the colon starting from cursor position 7.

**MR 2814**

**Indirect cursor position**

This module register specifies the cursor position, if **0** has been programmed as parameter.

| **Module register properties** | |
|---|---|
| Values | 0 ... Number of characters on the screen |
| Value after reset | 0 |

**Cursor position of HMIs**

The following table shows the correlation between cursor position as parameter of an instruction and the position on the screen.

| Designation | Line | Cursor position |
|---|---|---|
| LCD 16, LCD 110 | 1 | 1 ... 20 |
| | 2 | 21 ... 40 |
| | 3 | 41 ... 60 |
| | 4 | 61 ... 80 |
| LCD 23, LCD 27 | 1 | 1 ... 24 |
| LCD 34 | 2 | 25 ... 48 |
| LCD 52, LCD 54(Z) | 1 | 1 ... 16 |
| | 2 | 17 ... 32 |
| | 3 | 33 ... 48 |
| | 4 | 49 ... 64 |
| LCD 60 | 1 | 1 ... 40 |
| | 2 | 41 ... 80 |

# Setting the length of the input field

**Setting the length**

The input field length for numerical values is set in MR 2813.

```
MR 2813 = Number of figures + sign [+ decimal point]
```

**Example:**

| | |
|---|---|
| Number of figures: | 6 |
| Decimal point (MR 2811 = 0): | None |
| Field length (MR 2813): | 7 |

**MR 2813**

**Field length for UserInput**

This module register contains the length of the input field.

| **Module register properties** | |
|---|---|
| Values | 1 ... 12 |
| Value after reset | 11 |
| Takes effect | At the next STX instruction `UserInput()` |

# Setting the maximum number of decimal places

**Setting the number of decimal places**

The maximum number of decimal places is set in MR 2811.

If you enter decimal places, you might have to adjust the length of the display field in MR 2813.

Reason: The decimal point requires one place in the display field.

**MR 2811**

**Setting the maximum number of decimal places for UserInput instruction**

This module register specifies the maximum number of decimal places when inputting numerical values.

| **Module register properties** | |
|---|---|
| Values | 0 ... 4 |
| Value after reset | 4 |
| Takes effect | At the next STX instruction `UserInput()` |

# Setting the suggested value

**Setting the suggested value**

The suggested value for the STX instruction `UserInput()` is set in MR 2815.

**Displaying the suggested value**

The controller displays the suggested value for the STX instruction `UserInput()` using the same format settings as they are used for displaying numerical values.

**MR 2815**

**Suggested value for UserInput**

This module register specifies the suggested value which is displayed when the STX instruction `UserInput()` is invoked and after pressing the clear key **[C]**.

| **Module register properties** | |
|---|---|
| Values (Int) | -2,147,483,648 ... 2,147,483,647 |
| Values (Float) | $+/-(1.2x10^{-38}$ ... $3.4x10^{38})$ |
| Type | Int or Float depending on the value entered last |
| Takes effect | At the next STX instruction `UserInput()` |

# Polling the number of decimal places

**Polling the number of decimal places**

The number of decimal places which have been input can be read out from MR 2808.

**MR 2808**

**Number of decimal places which have been input in the case of UserInput instruction**

This module register specifies the number of decimal places which have been input by the user.

| Module register properties | |
|---|---|
| Values | 0 ... [MR 2811] |

# UserInput - Polling the status

**Polling the status**

The status of the `UserInput()` instruction can be polled from MR 2817.

**MR 2817**

**UserInput status**

This module register specifies the status of the `UserInput()` instruction.

| Module register properties | | |
|---|---|---|
| Values | 0 | No UserInput active |
| | 1 | UserInput active |

# UserInput - Aborting the instruction

**Aborting the active instruction**

By writing value **0** to MR 2817, you abort an active STX instruction `UserInput()`.

**Operating principle**

To abort an active STX instruction `UserInput()`, the controller proceeds as follows:

| Step | Description | |
|---|---|---|
| **1** | The controller disables the blinking cursor on the HMI. | |
| **2** | The controller throws the exception *USER_INPUT_BREAK*. | |
| **3** | **If ...** | **... then ...** |
| | ... an exception handling has been programmed, | ... the task proceeds with the exception handling. |
| | ... no exception handling has been programmed, | ... the task is aborted and the error is displayed in the error register. |

**Result:** The variable, which the result of the function is to be assigned to, will not be changed.

**MR 2817**

**UserInput status**

This module register specifies the status of the `UserInput()` instruction. By writing value **0** to MR 2817, you abort an active STX instruction `UserInput()`.

| **Module register properties** | | |
|---|---|---|
| Reading values | 0 | No UserInput active |
| | 1 | UserInput active |
| Writing values | 0 | Aborting UserInput |

**How to use this instruction**

```
Try
    Value := UserInput (0, 25);
Catch USER_INPUT_BREAK:
    Trace ('UserInput aborted !!');
End_Try;
```

# 10.9.7 Querying the keys

**Introduction**

This chapter gives a description on how HMI keys can be queried by the controller.

**Mapping keys**

The controller maps the keys of the HMIs to the following variables:

- Special flags
- Bits in registers which are overlaid to special flags

Flags and register bits assume the following states:

| Key | Special flag/register bit |
|---|---|
| Pressed | TRUE/1 |
| Not pressed | FALSE/0 |

**Restrictions**

While inputting numerical values, the keys used for this are not mapped to the key flags and register bits.

**Keys to be used for entering numerical values**

| Key | Description |
|---|---|
| **[0] ... [9]** | Entering a numerical value |
| **[.] or [,]** | Entering a decimal point or decimal comma |
| **[-]** | Entering a negative/positive numerical value;<br>you can press the key any time during input. |
| **[C]** | Clearing previous entries;<br>Displaying a suggested value a second time |
| **ENTER ([↵])** | Terminating the input process; accepting the value |

**Contents**


| Topic | Page |
|---|---|
| Assigning keys | 372 |
| Registers of basic flag numbers | 376 |

# Assigning keys

**Introduction**

HMI keys are assigned to an array of special flags and registers overlaid over them.

**Flag numbers**

Key flags are addressed relative to a basic flag number. Multi-display mode lets you set this basic flag number via registers.

| HMI | Register | Standard basic flag number | Flag numbers |
|---|---|---|---|
| Single display | - | 2000 | 2160 ... 2223 |
| 1 | MR 2829 | 2000 | 2160 ... 2223 |
| 2 | MR 2830 | 2000 | 2160 ... 2223 |
| 3 | MR 2831 | 2000 | 2160 ... 2223 |
| 4 | MR 2832 | 2000 | 2160 ... 2223 |

**Assignment**

Here, the assignments between keys, special flags and overlaid registers are listed. This assignment list applies to single-display mode and default settings of multi-display mode.

| Numerical keys | | | |
|---|---|---|---|
| **Key** | **Flag** | **Register bit (16-bit overlaying)** | **Register bit (32-bit overlaying)** |
| [0] | 2160 | 203139.0 | 203127.16 |
| [1] | 2161 | 203139.1 | 203127.17 |
| [2] | 2162 | 203139.2 | 203127.18 |
| [3] | 2163 | 203139.3 | 203127.19 |
| [4] | 2164 | 203139.4 | 203127.20 |
| [5] | 2165 | 203139.5 | 203127.21 |
| [6] | 2166 | 203139.6 | 203127.22 |
| [7] | 2167 | 203139.7 | 203127.23 |
| [8] | 2168 | 203139.8 | 203127.24 |
| [9] | 2169 | 203139.9 | 203127.25 |
| [SHIFT]+[0] | 2170 | 203139.10 | 203127.26 |
| [SHIFT]+[1] | 2171 | 203139.11 | 203127.27 |
| [SHIFT]+[2] | 2172 | 203139.12 | 203127.28 |
| [SHIFT]+[3] | 2173 | 203139.13 | 203127.29 |
| [SHIFT]+[4] | 2174 | 203139.14 | 203127.30 |
| [SHIFT]+[5] | 2175 | 203139.15 | 203127.31 |
| [SHIFT]+[6] | 2176 | 203140.0 | 203128.0 |
| [SHIFT]+[7] | 2177 | 203140.1 | 203128.1 |

| Numerical keys | | | |
|---|---|---|---|
| **Key** | **Flag** | **Register bit (16-bit overlaying)** | **Register bit (32-bit overlaying)** |
| [SHIFT]+[8] | 2178 | 203140.2 | 203128.2 |
| [SHIFT]+[9] | 2179 | 203140.3 | 203128.3 |

| Function keys | | | |
|---|---|---|---|
| **Key** | **Flag** | **Register bit (16-bit overlaying)** | **Register bit (32-bit overlaying)** |
| [F1] | 2201 | 203141.9 | 203128.25 |
| [F2] | 2202 | 203141.10 | 203128.26 |
| [F3] | 2203 | 203141.11 | 203128.27 |
| [F4] | 2204 | 203141.12 | 203128.28 |
| [F5] | 2205 | 203141.13 | 203128.29 |
| [F6] | 2206 | 203141.14 | 203128.30 |
| [F7] | 2207 | 203141.15 | 203128.31 |
| [F8] | 2208 | 203142.0 | 203129.0 |
| [F9] | 2209 | 203142.1 | 203129.1 |
| [F10] | 2210 | 203142.2 | 203129.2 |
| [F11] | 2211 | 203142.3 | 203129.3 |
| [F12] | 2212 | 203142.4 | 203129.4 |
| [SHIFT]+[F1] | 2181 | 203140.5 | 203128.5 |
| [SHIFT]+[F2] | 2182 | 203140.6 | 203128.6 |
| [SHIFT]+[F3] | 2183 | 203140.7 | 203128.7 |
| [SHIFT]+[F4] | 2184 | 203140.8 | 203128.8 |
| [SHIFT]+[F5] | 2185 | 203140.9 | 203128.9 |
| [SHIFT]+[F6] | 2186 | 203140.10 | 203128.10 |
| [SHIFT]+[F7] | 2187 | 203140.11 | 203128.11 |
| [SHIFT]+[F8] | 2188 | 203140.12 | 203128.12 |
| [SHIFT]+[F9] | 2189 | 203140.13 | 203128.13 |
| [SHIFT]+[F10] | 2190 | 203140.14 | 203128.14 |
| [SHIFT]+[F11] | 2191 | 203140.15 | 203128.15 |
| [SHIFT]+[F12] | 2192 | 203141.0 | 203128.16 |

| Special keys (do not apply to LCD 27) | | | |
|---|---|---|---|
| **Key** | **Flag** | **Register bit (16-bit overlaying)** | **Register bit (32-bit overlaying)** |
| [SHIFT]+[←] | 2193 | 203141.1 | 203128.17 |
| [SHIFT]+[→] | 2194 | 203141.2 | 203128.18 |
| [SHIFT]+[R] | 2195 | 203141.3 | 203128.19 |
| [SHIFT]+[I/O] | 2196 | 203141.4 | 203128.20 |
| [SHIFT]+[=] | 2197 | 203141.5 | 203128.21 |
| [SHIFT]+[C] | 2198 | 203141.6 | 203128.22 |
| [SHIFT]+ [ENTER] ([↵]) | 2199 | 203141.7 | 203128.23 |
| [SHIFT] | 2200 | 203141.8 | 203128.24 |
| [→] | 2213 | 203142.5 | 203129.5 |
| [←] | 2214 | 203142.6 | 203129.6 |
| [R] | 2215 | 203142.7 | 203129.7 |
| [I/O] | 2216 | 203142.8 | 203129.8 |
| [=] | 2217 | 203142.9 | 203129.9 |
| [C] | 2218 | 203142.10 | 203129.10 |
| [ENTER] ([↵]) | 2219 | 203142.11 | 203129.11 |
| [-] | 2220 | 203142.12 | 203129.12 |
| [SHIFT]+[-] | 2221 | 203142.13 | 203129.13 |
| [.] | 2222 | 203142.14 | 203129.14 |
| [SHIFT]+[.] | 2223 | 203142.15 | 203129.15 |

| LCD 27 | | | |
|---|---|---|---|
| **Key** | **Flag** | **Register bit (16-bit overlaying)** | **Register bit (32-bit overlaying)** |
| [↑] | 2209 | 203142.1 | 203129.1 |
| [↓] | 2210 | 203142.2 | 203129.2 |
| [C] | 2211 | 203142.3 | 203129.3 |
| [↵] | 2212 | 203142.4 | 203129.4 |

| NUM 25 | | | |
|---|---|---|---|
| **Key** | **Flag** | **Register bit (16-bit overlaying)** | **Register bit (32-bit overlaying)** |
| [S1] | 2206 | 203141.14 | 203128.30 |
| [S2] | 2207 | 203141.15 | 203128.31 |
| [S3] | 2208 | 203142.0 | 203129.0 |
| [S4] | 2209 | 203142.1 | 203129.1 |
| [S5] | 2210 | 203142.2 | 203129.2 |
| [SHIFT]+[S1] | 2186 | 203140.10 | 203128.10 |
| [SHIFT]+[S2] | 2187 | 203140.11 | 203128.11 |
| [SHIFT]+[S3] | 2188 | 203140.12 | 203128.12 |
| [SHIFT]+[S4] | 2189 | 203140.13 | 203128.13 |
| [SHIFT]+[S5] | 2190 | 203140.14 | 203128.14 |

# Registers of basic flag numbers

**Introduction**

Multi-display mode lets you set the basic flag numbers for HMI keys via registers.

**MR 2829**

## Basic flag number for device 1

This module register contains the basic flag number for HMI # 1.

| Module register properties | |
|---|---|
| Values | -160 ... 2080 |
| Value after reset | 2000 |
| Takes effect | On the next operation of a key |

**MR 2830**

## Basic flag number for device 2

This module register contains the basic flag number for HMI # 2.

| Module register properties | |
|---|---|
| Values | -160 ... 2080 |
| Value after reset | 2000 |
| Takes effect | On the next operation of a key |

**MR 2831**

## Basic flag number for device 3

This module register contains the basic flag number for HMI # 3.

| Module register properties | |
|---|---|
| Values | -160 ... 2080 |
| Value after reset | 2000 |
| Takes effect | On the next operation of a key |

**MR 2832**

## Basic flag number for device 4

This module register contains the basic flag number for HMI # 4.

| Module register properties | |
|---|---|
| Values | -160 ... 2080 |
| Value after reset | 2000 |
| Takes effect | On the next operation of a key |

# 10.9.8 Activating/deactivating LEDs

**Introduction**

This chapter gives a description on how you can activate or deactivate LEDs located in HMI keys.

**Mapping LEDs**

The controller reads out the state of LEDs located in HMI keys from the least significant 12 bits of the corresponding register.

| Register bit | LED |
|---|---|
| 1 | ON |
| 0 | OFF |

**Contents**


| Topic | Page |
|---|---|
| Assigning LEDs | 379 |
| Registers of LED register numbers | 380 |

# Assigning LEDs

**Introduction**

By default, LEDs located in HMI keys are assigned to a register which is overlaid by special flags.

**Register/flag numbers**

You can set the number of the register from which the LED state is read out via registers in multi-display mode.

| HMI | Register | Default LED register number | Flag numbers |
|---|---|---|---|
| Single display | - | 203143 | 2224 ... 2235 |
| 1 | MR 2833 | 203143 | 2224 ... 2235 |
| 2 | MR 2834 | 203143 | 2224 ... 2235 |
| 3 | MR 2835 | 203143 | 2224 ... 2235 |
| 4 | MR 2836 | 203143 | 2224 ... 2235 |

**Assignment**

Here, the assignments between keys, special flag and overlaid register are listed. This assignment list applies to single-display mode and default settings of multi-display mode.

| LED in the key | Flag | Register bit |
|---|---|---|
| [F1] | 2224 | 203143.0 |
| [F2] | 2225 | 203143.1 |
| [F3] | 2226 | 203143.2 |
| [F4] | 2227 | 203143.3 |
| [F5] | 2228 | 203143.4 |
| [F6] | 2229 | 203143.5 |
| [F7] | 2230 | 203143.6 |
| [F8] | 2231 | 203143.7 |
| [F9] | 2232 | 203143.8 |
| [F10] | 2233 | 203143.9 |
| [F11] | 2234 | 203143.10 |
| [F12] | 2235 | 203143.11 |

# Registers of LED register numbers

**Introduction**

In multi-display mode, the register numbers indicating the state of LEDs in HMIs can be set via registers.

**MR 2833**

**LED register number for device # 1**

This module register contains the LED register number for HMI # 1.

| Module register properties | |
|---|---|
| Values | 100000 ... 1059999 |
| Value after reset | 203143 |

**MR 2834**

**LED register number for device # 2**

This module register contains the LED register number for HMI # 2.

| Module register properties | |
|---|---|
| Values | 100000 ... 1059999 |
| Value after reset | 203143 |

**MR 2835**

**LED register number for device # 3**

This module register contains the LED register number for HMI # 3.

| Module register properties | |
|---|---|
| Values | 100000 ... 1059999 |
| Value after reset | 203143 |

**MR 2836**

## LED register number for device # 4

This module register contains the LED register number for HMI # 4.

| Module register properties | |
|---|---|
| Values | 100000 ... 1059999 |
| Value after reset | 203143 |

## 10.9.9 Monitor functions

**Introduction**

This chapter provides a description on how an HMI can be used to display and change variables independent of the application program.

**Prerequisites**

For the monitor functions, the following requirements have to be met:

- An HMI with numeric keypad must be connected to the controller.
- The monitor functions must not be blocked in the configuration registers.
- The STX instruction `UserInput()` for inputting numerical values must not be active.

**Restrictions**

The monitor function can only access controller variables which are assigned to permanent addresses.

- Registers (%VL)
- Flags (%MX)
- Inputs (%IX)
- Outputs (%QX)

**Multi-display mode**

In multi-display mode, the following must be taken into account:

- The monitor function is only displayed on the HMI on which you have activated it by pressing the key **[R]** or the key **[I/O]**.
- The controller is not able to tell apart on which HMI subsequent keystrokes are carried out.

**Contents**


| Topic | Page |
|---|---|
| Overview of displays and HMIs | 383 |
| Meaning of keys in monitor function | 384 |
| Displaying and changing variables | 385 |
| Configuring the monitor function | 387 |

# Overview of displays and HMIs

**HMIs supporting the monitor function**

The following table lists the alphanumeric displays and HMIs by Jetter AG which also feature the monitor function.

| Designation | Keys | Variables |
|---|---|---|
| **LCD 16 + NUM 25** | [R] and [I/O] | ■ Registers<br>■ Flags<br>■ Inputs<br>■ Outputs |
| **LCD 34** | [R] | ■ Registers<br>■ Flags |
| **LCD 52** | [R] and [I/O] | ■ Registers<br>■ Flags<br>■ Inputs<br>■ Outputs |
| **LCD 54(Z)** | [R] and [I/O] | ■ Registers<br>■ Flags<br>■ Inputs<br>■ Outputs |
| **LCD 60** | [R] and [I/O] | ■ Registers<br>■ Flags<br>■ Inputs<br>■ Outputs |
| **LCD 110** | [R] and [I/O] | ■ Registers<br>■ Flags<br>■ Inputs<br>■ Outputs |

# Meaning of keys in monitor function

**Keys being used in monitor function**

The following keys are used in monitor function:

| Key | Description |
|---|---|
| **[R]** | Initiating the monitor function for registers or flags |
| **[I/O]** | Initiating the monitor function for outputs, inputs or flags |
| **[0] ... [9]** | Entering a variable number or a decimal value |
| **[.] or [,]** | Entering a decimal point or decimal comma |
| **[-]** | Entering a negative/positive numerical value;<br>you can press the key any time during input. |
| **[C]** | ▪ Clearing previous entries<br>▪ Aborting the monitor function |
| **ENTER ([↵])** | ▪ Completing the input process; displaying the variable or accepting the value<br>▪ Switching between monitor screen and normal display |

**Restrictions**

While the monitor function is active, keys used for this function are not mapped to the key flags.

## Displaying and changing variables

**Initiating the monitor function**

To initiate the monitor function, press the key **[R]** or the key **[I/O]**. You are able to change te variable type using these keys as long as the entry of the variable number has not been completed yet by pressing the key **ENTER** ([↵]).

**Displaying register content**

To display a register content, proceed as follows:

| Step | Action |
|---|---|
| **1** | Press the key **[R]**.<br>**Result:** The HMI switches to displaying the monitor function. |
| **2** | Enter the register number. |
| **3** | Press the key **ENTER** ([↵]). |

**Result:** The register content is displayed for the period of time set in MR 2819 *Monitor functions - Display duration*. Once this period has elapsed, the HMI displays the normal screen.

**Displaying the flag state**

To display a flag status, proceed as follows:

| Step | Action |
|---|---|
| **1** | Press the key **[R]** twice or the key **[I/O]** three times.<br>**Result:** The HMI switches to displaying the monitor function. |
| **2** | Enter the flag number. |
| **3** | Press the key **ENTER** ([↵]). |

**Result:** The flag status is displayed for the period of time set in MR 2819 *Monitor functions - Display duration*. Once this period has elapsed, the HMI displays the normal screen.

**Displaying an output state**

To display an output state, proceed as follows:

| Step | Action |
|---|---|
| **1** | Press the key **[I/O]**.<br>**Result:** The HMI switches to displaying the monitor function. |
| **2** | Enter the output number. |
| **3** | Press the key **ENTER** ([↵]). |

**Result:** The output state is displayed for the period of time set in MR 2819 *Monitor functions - Display duration*. Once this period has elapsed, the HMI displays the normal screen.

**Displaying an input state**

To display an input state, proceed as follows:

| Step | Action |
|---|---|
| **1** | Press the key **[I/O]** twice.<br>**Result:** The HMI switches to displaying the monitor function. |
| **2** | Enter the input number. |
| **3** | Press the key **ENTER** ([↵]). |

**Result:** The input state is displayed for the period of time set in MR 2819 *Monitor functions - Display duration*. Once this period has elapsed, the HMI displays the normal screen.

**Modifying a variable value**

To modify a variable value, proceed as follows:

| Step | Action |
|---|---|
| **1** | Have the value of the variable displayed (see above). |
| **2** | Press the key **[=]**.<br>**Result:** You are prompted to enter a new value for this variable. Meanwhile, the current value is displayed. |
| **3** | Enter a new value. |
| **4** | Press the key **ENTER** ([↵]). |

**Result:** The new value is written to the variable. The variable value is displayed for the period of time set in MR 2819 *Monitor functions - Display duration*. Once this period has elapsed, the HMI displays the normal screen.

**Aborting the display function**

If you wish to abort the display of a variable before the set time (default: 3.5 sec) has elapsed and return to the normal screen, press the **ENTER** ([↵]) key.

**Redisplaying a variable**

If the normal display is active again and you wish to have the variable value redisplayed, press the **ENTER** ([↵]) key. The monitor display will be re-activated for 3.5 seconds.

# Configuring the monitor function

**Introduction**

The following registers are for configuring the monitor functions.

**MR 2818**

## Disabling/enabling monitor functions

This module register is bit-coded. These bits can be used to disable/enable individual monitor functions. Keys are also mapped to flags with the monitor function disabled.

| **Module register properties** | |
|---|---|
| Values | 0 ... 255 |
| Value after reset | 255 |

| **Meaning of the individual bits** | |
|---|---|
| **Bit 0** | **Taste [R]** |
| | 0 = Key **[R]** has no monitor function. |
| | 1 = Key **[R]** has got a monitor function. |
| **Bit 1** | **Displaying the flag state** |
| | 0 = Keys **[R]** and **[I/O]** without monitor function **Display flag state** |
| | 1 = Keys **[R]** and **[I/O]** with monitor function **Display flag state** |
| **Bit 2** | **Displaying an output state** |
| | 0 = Key **[I/O]** without monitor function **Display output state** |
| | 1 = Key **[I/O]** with monitor function **Display output state** |
| **Bit 3** | **Displaying an input state** |
| | 0 = Key **[I/O]** without monitor function **Display input state** |
| | 1 = Key **[I/O]** with monitor function **Display input state** |
| **Bit 4** | **Changing register contents** |
| | 0 = Key **[=]** without monitor function **Change register contents** |
| | 1 = Key **[=]** with monitor function **Change register contents** |
| **Bit 5** | **Changing the flag state** |
| | 0 = Key **[=]** without monitor function **Change flag state** |
| | 1 = Key **[=]** with monitor function **Change flag state** |
| **Bit 6** | **Changing an output state** |
| | 0 = Key **[=]** without monitor function **Change output state** |
| | 1 = Key **[=]** with monitor function **Change output state** |
| **Bit 7** | **Permanent indication of input state** |
| | 0 = Key **[=]** without monitor function |
| | 1 = Key **[=]** with monitor function |

**MR 2819** **Display time for monitor functions**

This module register contains the display time in multiples of 100 ms.

| **Module register properties** | |
|---|---|
| Values | 0 ... 65,535 |
| Value after reset | 35 (3.5 s) |
| Takes effect | Next time when the display changes to monitor operation |

**MR 2820** **Switch-over to monitor display**

This module register is for configuring the function of the key **ENTER** ([↵]).

| **Module register properties** | | |
|---|---|---|
| Values | 0 | Switching between monitor screen and normal display is activated. |
| | 1 | Switching between monitor screen and normal display is deactivated. |

**MR 2821** **Dialog language**

This module register is for configuring the dialog language used for the monitor function.

| **Module register properties** | | |
|---|---|---|
| Values | 0 | German |
| | 1 | English |
| Takes effect | Next time the monitor function is launched | |

# 10.10 Controlling printer and serial interfaces

**Introduction**

This chapter contains information on how to control printer and serial interfaces from within the application program in a JC-360.

**Controlling the interfaces**

Printer and serial interfaces can be controlled in two ways:

- Direct access to registers of the interface
- Using display functions included in the STX language

**Direct access to the interface**

In order to output special or control characters or to retrieve the status of the external device, direct access to the registers of the interface must be used. For more information on how to access registers refer to the documentation on the corresponding module.

**Display functions**

This chapter describes how to control the interfaces using display functions. It also describes the registers used to parameterize the display functions. The controller provides the following display functions:

- Displaying texts
- Displaying the contents of variables

**Requirements**

In this manual we proceed from the assumption that the user is familiar with the following STX instructions: `DisplayText()`, `DisplayText2()`, and `DisplayValue()`. For a more detailed description of the instructions mentioned above refer to the online help included in the programming tool JetSym.

**Contents**


| Topic | Page |
|---|---|
| Supported serial interfaces | 390 |
| Registers | 392 |
| Module numbers - Interface modules | 395 |
| Outputting texts | 397 |
| Outputting numerical values | 401 |

# 10.10.1 Supported serial interfaces

**Introduction**

This chapter lists the printers and serial interfaces which are supported by the JC-360.

**Contents**


| Topic | Page |
|---|---|
| Overview - Interfaces | 391 |

# Overview - Interfaces

**List of printers and serial interfaces**

The following table lists the supported printers and serial interfaces. It also indicates the device number which must be addressed by the display instruction in order to output information on the interface.

| Module | Interface | Device number |
|---|---|---|
| JX2-PRN1 | Centronics printer module | 8 |
| User-programmable serial interface | Serial interface of the CPU | 9 |
| JX2-SER1 | Serial interface module | 11 |
| JX3-MIX2 | Serial interface on the module | 11 |

**Configuring interfaces**

For more information on how to configure and program interfaces refer to the documentation on the corresponding module.

| Module | Documentation |
|---|---|
| JX2-PRN1 | jx2_prn1_ba_xxxx_manual.pdf |
| User-programmable serial interface | *User-programmable serial interface* (see page 477) |
| JX2-SER1 | jx2_ser1_ba_xxxx_manual.pdf |
| JX3-MIX2 | jx3_mix2_ba_xxxx_manual.pdf |

# 10.10.2 Registers

**Introduction**

This chapter provides you with an overview of registers which are used to parameterize printer and serial interfaces.

**Restrictions**

The settings made in the given registers are of global effect, that is, they will impact all functions for controlling printers and serial interfaces. If different settings are used in several tasks of the application program, then these settings may impact each other.

**Contents**


| Topic | Page |
|---|---|
| Register numbers | 393 |
| Registers - Overview | 394 |

# Register numbers

**Introduction**

The registers are combined into one register block. The basic register number of this block is dependent on the controller.

**Register numbers**

| Device | Basic register number | Register numbers |
|---|---|---|
| JC-360 | 220000 | 222806 ... 222838 |

**Determining register numbers**

In this chapter, only the last four figures of a register number are specified. e.g. MR 2838. Add to this module register number the basic register number of the corresponding device to determine the complete register number, for example 222838.

# Registers - Overview

**Registers - Overview**

The following table provides you with an overview of existing registers. For a detailed description of these registers refer to the following chapters.

| Registers | Description |
|---|---|
| **MR 2806** | Text selection (`DisplayText2()`) |
| **MR 2810** | Number of decimal places (`DisplayValue()`) |
| **MR 2812** | Field length (`DisplayValue()`) |
| **MR 2816** | Displaying the sign |
| **MR 2824** | Indirect device number -<br>Device number of default device |
| **MR 2837** | Module number - Printer module |
| **MR 2838** | Module number - serial interface module |

# 10.10.3 Module numbers - Interface modules

**Introduction**

To be able to redirect display instructions to a printer or serial interface module connected to the JX2 or JX3 system bus, the module number must be set.

Redirection to an internal user-programmable serial interface is clearly defined by the device number. Therefore, no configuration effort is required.

**Contents**


| Topic | Page |
|---|---|
| Configuring module numbers | 396 |

# Configuring module numbers

**Determining module numbers**

The module number to be entered is calculated based on the number of the module on the system bus plus a constant value considering the system bus:

```
Module number = number of the module + system bus constant
```

| System bus | System bus constant |
|---|---|
| JX3 | 100 |
| JX2 | 200 |

**MR 2837**

## Module number - Printer module

This module register holds the number of the module which the display instruction is redirected to (device # 8).

| Module register properties | |
|---|---|
| Values (JX3 bus) | 102 ... 117 |
| Values (JX2 bus) | 202 ... 224 |
| Takes effect | Next time when STX instruction `DisplayText()` or `DisplayValue()` is issued |

**MR 2838**

## Module number - Serial interface module

This module register holds the number of the module which the display instruction is redirected to (device # 11).

| Module register properties | |
|---|---|
| Values (JX3 bus) | 102 ... 117 |
| Values (JX2 bus) | 202 ... 224 |
| Takes effect | Next time when STX instruction `DisplayText()` or `DisplayValue()` is issued |

# 10.10.4 Outputting texts

**Introduction**

This chapter describes how to output texts via printer or serial interface and how to parameterize the corresponding STX instructions.

**STX instructions**

In order to output texts, use the following STX instructions (STX functions):

- `DisplayText()`
- `DisplayText2()`

**Contents**


| Topic | Page |
|---|---|
| STX instructions for outputting texts | 398 |
| Device numbers | 400 |

## STX instructions for outputting texts

**Function declaration**

```
Function DisplayText(Dev:Int,
                     Pos:Int,
                     Const Ref Text:String);
```

**Function parameters**

| Parameter | Value | Description |
|---|---|---|
| Dev | 8 ... 11 | Number of the device on which the text is to be output |
| Pos | Not relevant | Will not be evaluated |
| Text | Text to be output | Constant text, or name of a string variable |

**How to use this instruction**

To output a text on a printer module, the instruction must be invoked as follows:

```
DisplayText(8, 0, 'Hello World !');
DisplayText(8, 0, StringVar);
```

**Operating principle**

The first instruction causes the printer module to output the text 'Hello World!'. Then, the second STX instruction causes the content of the string variable **StringVar** to be output.

The task in the application program stops at the instruction `DisplayText()` until the whole text has been output.

**Function declaration**

```
Function DisplayText2(Dev:Int,
                      Pos:Int,
                      Const Ref Text1:String,
                      Const Ref Text2:String);
```

**Function parameters**

| Parameter | Value | Description |
|---|---|---|
| Dev | 8 ... 11 | Number of the device on which the text is to be output |
| Pos | Not relevant | Will not be evaluated |
| Text1 | Text to be output | Constant text, or name of a string variable |
| Text2 | Text to be output | Constant text, or name of a string variable |

**How to use this instruction**

To output one text out of two texts on a serial interface module, the instruction must be invoked as follows:

```
DisplayText2(11, 0, 'Fehler:', 'Error:');
```

**Operating principle**

The STX instruction causes either the text 'Fehler:', or 'Error:' to be output on a serial interface module. MR 2806 lets you control which text will be output.

The task in the application program stops at the instruction `DisplayText2()` until the whole text has been output.

**MR 2806**

## Text selection for DisplayText2

The value in this module register specifies which one of the two texts is to be output.

| Module register properties | | |
|---|---|---|
| Values | 0 | Text1 |
| | 1 | Text2 |

# Device numbers

**Introduction**

The device number lets you define the HMI.

**Device numbers**

You may enter the following values for the parameter Device number:

| Number | Element | Description |
|---|---|---|
| **0** | Default device | MR 2824 holds the number of the device to be used. |
| **1** | HMI # 1 | Multi-display mode |
| **2** | HMI # 2 | Single-/Multi-display mode |
| **3** | HMI # 3 | Multi-display mode |
| **4** | HMI # 4 | Multi-display mode |
| **5 ... 7** | Reserved | Do not use |
| **8** | Printer module | Output of data on a printer module connected to the JX2 or JX3 system bus |
| **9, 10** | Serial interface | Output of data on a user-programmable serial interface |
| **11** | Serial interface module | Output of data on a serial interface module connected to the JX2 or JX3 system bus |

**MR 2824**

**Device number of the default device**

This module register contains the device number of the default device. If you always specify the default device in the application program (device number = 0), you can select during runtime which device is actually to be used.

| **Module register properties** | |
|---|---|
| Values | 1 ... 11 |
| Value after reset | 2 |

# 10.10.5 Outputting numerical values

**Introduction**

This chapter describes how to output numerical values via printer or serial interface and how to parameterize the corresponding STX instructions. These numerical values may be constants or contents of registers and variables.

**STX instruction**

To display numerical values, use the following STX instruction:

- `DisplayValue()`

**Formatting the values to be output**

The format of numerical values to be output can be adapted to suit your needs. The following parameters can be adjusted:

- Display length
- Number of decimal places
- With or without sign
- Decimal or hexadecimal notation

**Outputting numerical values**

When outputting numerical values the following formatting rules apply:

- The numerical value is displayed right-aligned.
- The sign is the first character which is output if the sign has not been disabled before.
- The first numerical value which is output is the leading space or the leftmost figure.
- Positive values are preceded by a space character as sign. Negative values are preceded by "-".
- If the display field is too small, the leftmost figures are truncated.
- The value is rounded to the set decimal places.

**Contents**


| Topic | Page |
|---|---|
| STX instruction for outputting numerical values | 402 |
| Device numbers | 403 |
| Setting the length of the display field | 404 |
| Setting the sign option | 405 |
| Setting the number of decimal places | 406 |
| Setting the format of numerical values | 407 |

# STX instruction for outputting numerical values

**Function declaration**

```
Function DisplayValue(Dev:Int,
                      Pos:Int,
                      Value:Double);
```

**Function parameters**

| Parameter | Value | Description |
|---|---|---|
| Dev | 8 ... 11 | Number of the device where the value is to be output |
| Pos | Not relevant | Will not be evaluated |
| Value | Value to be output | Constant value, name of a register or a variable |

**How to use this instruction**

To output a numerical value on a printer module, the instruction must be invoked as follows:

```
DisplayValue(8, 0, -12.345);
DisplayText(8, 0, '$t');
DisplayValue(8, 0, Axis2.Position);
DisplayText(8, 0, '$n');
```

**Operating principle**

The first STX instruction lets you output value **-12,345**. The second STX instruction inserts a tab ($t). The third STX instruction lets you output the content of the variable **Axis2.Position**. Finally, the fourth STX instruction triggers a carriage return and a line feed ($n).

The task in the application program stops at the instruction `DisplayText()` or `DisplayValue()` until the whole text/value string has been output.

# Device numbers

**Introduction**

The device number lets you define the HMI.

**Device numbers**

You may enter the following values for the parameter Device number:

| Number | Element | Description |
|---|---|---|
| **0** | Default device | MR 2824 holds the number of the device to be used. |
| **1** | HMI # 1 | Multi-display mode |
| **2** | HMI # 2 | Single-/Multi-display mode |
| **3** | HMI # 3 | Multi-display mode |
| **4** | HMI # 4 | Multi-display mode |
| **5 ... 7** | Reserved | Do not use |
| **8** | Printer module | Output of data on a printer module connected to the JX2 or JX3 system bus |
| **9, 10** | Serial interface | Output of data on a user-programmable serial interface |
| **11** | Serial interface module | Output of data on a serial interface module connected to the JX2 or JX3 system bus |

**MR 2824**

**Device number of the default device**

This module register contains the device number of the default device. If you always specify the default device in the application program (device number = 0), you can select during runtime which device is actually to be used.

| **Module register properties** | |
|---|---|
| Values | 1 ... 11 |
| Value after reset | 2 |

# Setting the length of the display field

**Setting the length**

MR 2812 lets you set the length of the display field for numerical values.

```
MR 2812 = Number of figures + sign [+ decimal point]
```

**Example:**

| | |
|---|---|
| Number of figures: | 6 |
| Sign (MR 2816): | 0 (yes) |
| Decimal point: | None |
| Field length (MR 2812): | 7 |
| Display: | 7 characters |

**MR 2812**

**Field length for DisplayValue**

This module register holds the length of the display field.

| **Module register properties** | |
|---|---|
| Values | 1 ... 12 |
| Value after reset | 11 |
| Takes effect | Next time when STX instruction `DisplayValue()` is issued |

# Setting the sign option

**Setting the sign option**

MR 2816 lets you set whether the sign is displayed or not.

In setting the display field length in MR 2812 continue to add the sign's place, even if no sign is to be displayed.

**Example:**

| | |
|---|---|
| Number of figures: | 6 |
| Sign (MR 2816): | 1 (no) |
| Decimal point: | None |
| Field length (MR 2812): | 7 |
| Display: | 6 characters |

**MR 2816**

**Displaying the sign**

| **Module register properties** | | |
|---|---|---|
| Values | 0 | Sign will be displayed. |
| | 1 | Sign will not be displayed. |
| Takes effect | Next time when STX instruction `DisplayValue()` is issued | |

# Setting the number of decimal places

**Setting the number of decimal places**

The number of decimal places is set in MR 2810.

If you add decimal places, you might have to adjust the length of the display field in MR 2812.

Reason: The decimal point requires one place in the display field.

**MR 2810**

**Number of decimal places for DisplayValue instructions**

This module register holds the number of decimal places when displaying numerical values.

| **Module register properties** | |
|---|---|
| Values | 0 ... 4 |
| Takes effect | Next time when STX instruction `DisplayValue()` is issued |

# Setting the format of numerical values

**Setting the format of numerical values**

In setting the format of numerical values you have the choice between decimal or hexadecimal format. Flag 2060 lets you set the format of numerical values.

**Flag 2060**

**Format of numerical values**

| **Flag properties** | | |
|---|---|---|
| Values | 0 | Decimal |
| | 1 | Hexadecimal |
| Takes effect | Next time when STX instruction `DisplayValue()` is issued | |

# 10.11 JX2 system bus

**Introduction**

The controller JC-360 features an internal JX2 system bus master. Several modules can be connected to the JX2 system bus. Access to these modules is completely transparent.

**Configuration**

The JX2 system bus needs not be configured. The JC-360 is able to automatically detect and commission connected modules.

Only the baud rate must be set by the user.

**Connectable modules**

- JX2-I/O modules
- JX2 slave modules
- Servo amplifiers JetMove 1xx, JetMove 2xx, and JetMove 6xx
- IP67-I/O modules LioN-S and LJX7-CSL
- Third-party CANopen® modules, e.g. valve terminals

**Contents**


| Topic | Page |
|---|---|
| Module array and module codes of connected modules | 409 |
| JX2 system bus - Baud rate | 412 |
| Dummy modules on the JX2 system bus | 414 |
| Monitoring intervals on the JX2 system bus | 415 |
| JX2 system bus - Description of non-volatile registers | 417 |
| Register description of modules connected to the JX2 system bus | 421 |
| Register description - Error logging on the JX2 system bus | 423 |
| Register description - Timeout and interval times for modules on the JX2 system bus | 426 |
| Register description - Retry counter for JX2 system bus modules | 429 |
| Register description - Versions of JX2 system bus drivers | 430 |

# Module array and module codes of connected modules

**Module array**

A unique module code is assigned to each module connected to the JX2 system bus. The JC-360 enters all modules detected during initialization into the module array.

Modules are entered into the module array in the following order:

- First, JX2-I/O, JX2 slave modules, and JetMoves are are entered into the module array according to their position on the JX2 system bus. Configured dummy modules are entered here, too.
- Then, IP67-I/O modules LioN-S and LJX7-CSL are entered according to the set module number. Vacant positions between module numbers are filled with dummy modules.
- Finally, CANopen® modules are entered according to their set module number. Vacant positions between module numbers are also filled with dummy modules.

**Registers - Overview**

| Registers | Description |
|---|---|
| **R 200002015** | Index to module array |
| **R 200002016** | Module array |

**Reading the module array**

The module array is accessed indirectly.

| Step | Action |
|---|---|
| **1** | Enter the number of the module in the module array into R 200002015 *Index to Module Array*. |
| **2** | Read out the module code from R 200002016 *Module Array*. |

**Module codes of JX2-I/O modules**

| Module code | Module | Description |
|---|---|---|
| 0 | JX2-OD8 | 8 digital outputs |
| 1 | JX2-ID8 | 8 digital inputs |
| 2 | JX2-IO16 | 8 digital inputs and 8 digital outputs |
| 3 | JX2-IA4 | 4 analog inputs |
| 4 | JX2-OA4 | 4 analog outputs |
| 5 | JX2-CNT1 | Counter module |
| 6 | JX2-PRN1 | Printer Interface Module |
| 7 | JX2-SER1 | Module with serial interface |
| 9 | JX2-TP20-R | Module with operating keys |

**Module codes of LJX7-CSL modules**

| Module code | Module | Description |
|---|---|---|
| 10 | LJX7-CSL-108-ID16 | 16 digital inputs |
| 11 | LJX7-CSL-109-ID16-NPN | 16 digital NPN inputs |
| 12 | LJX7-CSL-107-OD8-2A | 8 digital outputs, 2 A |
| 13 | LJX7-CSL-114-OD16 | 16 digital outputs |
| 14 | LJX7-CSL-113-ID8-OD8 | 8 digital inputs and 8 digital outputs |

**Module codes of LioN-S modules**

| Module code | Module | Description |
|---|---|---|
| 50 | 0930 CSL 650 | 8 universal I/Os |
| 51 | 0930 CSL 651 | 8 digital inputs |

**Module codes of CANopen® modules**

| Module code | Module | Manufacturer |
|---|---|---|
| 64 | JX-SIO | Jetter AG |
| 65 | CPV-Direct | Festo AG & Co. |
| 66 | CPX terminal | Festo AG & Co. |
| 67 | Valve terminal type 8640 | Bürkert GmbH & Co. KG |
| 68 | I/O-SYSTEM 750 | WAGO Kontakttechnik GmbH |
| 69 | SI unit EX 120 | SMC Pneumatik GmbH |
| 70 | 8200 vector / 2175 | Lenze Drives Systems GmbH |
| 71 | SI unit EX 250 | SMC Pneumatik GmbH |
| 73 | CPX-Terminal, virtual | - |
| 75 | EPOS | maxon motor ag |
| 79 | BWU 1821 | Bihl+Wiedemann GmbH |
| 80 | 8200 vector / CANopen® PT | Lenze Drives Systems GmbH |
| 108 | SFC-LAC | Festo AG & Co. |
| 109 | SFC-LACI | Festo AG & Co. |
| 110 | SFC-DC | Festo AG & Co. |
| 111 | MTR-DCI | Festo AG & Co. |

**Module codes of CANopen® slave modules**

| Module code | Module | Manufacturer |
|---|---|---|
| 103 | Milan drive | GFC Antriebssysteme GmbH |
| 104 | EcoStep | Jenaer Antriebstechnik GmbH |

**Module codes of JX2 slave modules**

| Module code | Module | Description |
|---|---|---|
| 128 | JX2-SV1 | Servo controller module |
| 129 | CAN-DIMA | Servo amplifiers |
| 130 | JX2-SM2 | Stepper motor controller |
| 131 | JX2-SM1D | Stepper motor output stage |
| 132 | JX2-PID1 | PID controllers |
| 133 | JX2-PROFI1 | Profibus DP slave |
| 135 | JM-2xx series | Servo amplifiers |
| 137 | JM-D203 | Servo amplifier for 2 axes |
| 138 | JM-1xx series | Servo amplifiers |
| 146 | JM-6xx series | Servo amplifiers |

**Module codes - Dummy modules**

| Module code | Module | Description |
|---|---|---|
| 252 | CANopen® dummy module | - |
| 253 | JX2-Slave dummy module | - |
| 254 | I/O dummy module | - |
| 255 | Unknown module | - |

# JX2 system bus - Baud rate

**Introduction**

The user configures the baud rate of the JX2 system bus. A reduced baud rate allows longer lines.

**Registers - Overview**

The value contained in the register for baud rate configuration is remanent. Any changes become effective only after the controller JC-360 has been re-booted.

| Register | Description |
|---|---|
| **R 200002029** | Baud rate of JX2 system bus |

**Modules with automatic baud rate detection**

Most modules on the JX2 system bus will automatically detect the baud rate of the controller JC-360. The following modules feature automatic baud rate detection (Auto Baud Rate):

- JX2-I/O modules
- JX2 slave modules
- JetMove 1xx, 2xx und 6xx
- IP67-I/O modules LioN-S and LJX7-CSL

**Baud rate**

The baud rate setting depends on the number of modules connected to the JX2 system bus.

| JX2-I/O modules<br>JX2 slave modules<br>JetMove | JX-SIO<br>IP67-I/O modules<br>CANopen® modules | 1000 kBaud | 500 kBaud | 250 kBaud | 125 kBaud |
|---|---|---|---|---|---|
| x | | x | x | x | x |
| | x | x | x | x | x |
| x | x | x | | | x |

**Changing the Baud rate**

To set the baud rate of the JX2 system bus proceed as follows:

| Step | Action | |
|---|---|---|
| **1** | Adjust the baud rate of the controller in R 200002029 *Baud rate JX2 system bus*. | |
| | **If ...** | **... then ...** |
| | ... baud rate = 1,000 kBaud, | ... R 200002029 := 7. |
| | ... baud rate = 500 kBaud, | ... R 200002029 := 6. |
| | ... baud rate = 250 kBaud, | ... R 200002029 := 5. |
| | ... baud rate = 125 kBaud, | ... R 200002029 := 4. |
| **2** | Switch off the controller and all modules on the JX2 system bus. | |
| **3** | Adjust the baud rate on all modules without Auto Baud Rate. | |
| **4** | Switch on the controller and all modules on the JX2 system bus. | |

**Result:**

When initializing the JX2 system bus, the adjusted baud rate values were used.

# Dummy modules on the JX2 system bus

**Dummy modules**

Ein Dummy-Modul ist ein Modul am JX2-Systembus, das physikalisch nicht vorhanden ist. When assigning I/O and slave module numbers, the controller JC-360 treats dummy modules as if they were existing modules.

Dummy modules allow the user to insert a non-existent module between existing modules.

**Registers - Overview**

The value contained in the registers for dummy module configuration is remanent. Any changes become effective only after the controller JC-360 has been re-booted.

| Register | Description |
|---|---|
| **R 200002023** | I/O dummy modules |
| **R 200002024** | JX2-Slave dummy modules |

**Allowed dummy modules**

- JX2-I/O modules
- JX2 slave modules
- JetMove 1xx, 2xx und 6xx

**Modules with Address Selector**

Not all modules on the JX2 system bus can be configured as dummy module. Modules with address selector are configured by means of this address selector. The controller fills vacant positions between addresses in the module array with dummy modules.

**Configuring dummy modules**

| Step | Action |
|---|---|
| **1** | Modify the dummy module configuration within the controller via R 200002023 and R 200002024. |
| **2** | Switch the controller off. |
| **3** | Then, switch the controller on.<br>**Result:**<br>When initializing the JX2 system bus, the configured dummy modules were taken into account. Information from the dummy modules can be retrieved via module array. |

# Monitoring intervals on the JX2 system bus

**Introduction**

The JC-360 checks in regular intervals whether communication with the modules connected to the JX2 system bus is still working. To this end, it sends monitoring signals to the modules and waits for a response.

**Registers - Overview**

| Register | Description |
|---|---|
| **R 210004** | Error registers - JC-360 |
| **R 200002008** | Error registers - JX2 system bus |
| **R 200002011** | I/O module number at timeout |
| **R 200002028** | Cycle time of monitoring interval |
| **R 200002760** | Max. number of I/O update retries |

**Monitoring of JX2-I/O modules**

When monitoring JX2-I/O modules, a missing response will not immediately result in a timeout error on the JC-360.

| Step | Description | |
|---|---|---|
| **1** | The JC-360 sends a monitoring telegram to a JX2-I/O module.<br>The cycle time can be configured in R 200002028. | |
| **2** | **If ...** | **... then ...** |
| | ... a response has been received, | ... the JC-360 proceeds monitoring the next JX2-I/O module by taking step 1. |
| | ... no response has been received and the number of allowed retries has not been maxed out, | ... the JC-360 sends one more monitoring telegram. |
| | ... no response has been received and the number of allowed retries has been maxed out, | ... the JC-360 generates a timeout in step 3. |
| **3** | The JC-360 generates a timeout:<br>■ The I/O module number is entered into R 200002011.<br>■ In R 200002008 bit 3 is set to 1.<br>■ In R 210004 bit 2 is set to 1.<br>■ The red error LED on the JC-360 is lit. | |
| **4** | The JC-360 proceeds with step 1 for the next JX2-I/O module. | |

**Monitoring of IP67 and CANopen® modules**

When monitoring IP67 and CANopen® modules, a missing response will immediately result in a timeout error on the JC-360. IP67 and CANopen® modules are monitored using the CANopen® service **nodeguarding**.

| Step | Description | |
|---|---|---|
| **1** | The JC-360 sends a monitoring telegram to an IP67 or CANopen® module.<br>The cycle time can be configured in R 200002028. | |
| **2** | **If ...** | **... then ...** |
| | ... a response has been received, | ... the JC-360 proceeds with step 1 for monitoring the next module. |
| | ... no response has been received and the number of allowed retries has been maxed out, | ... the JC-360 generates a timeout in step 3. |
| **3** | The JC-360 generates a timeout:<br>■ The I/O module number is entered into R 200002011.<br>■ In R 200002008 bit 3 is set to 1.<br>■ In R 210004 bit 2 is set to 1.<br>■ The red error LED on the JC-360 is lit. | |
| **4** | The JC-360 proceeds with step 1 for monitoring the next IP67- oder CANopen® module. | |

# JX2 system bus - Description of non-volatile registers

**Introduction**

Non-volatile registers let you configure the JX2 system bus. Any changes to a non-volatile register become effective only after the JC-360 has been relaunched.

**R 200002023**

## I/O dummy modules

Each bit in this register represents an I/O module on the JX2 system bus.

| Meaning of the individual bits | | |
|---|---|---|
| **Bit 0** | **Configuration of I/O module 2** | |
| | 0 = | I/O module is a dummy module |
| | 1 = | I/O module is not a dummy module |
| **Bit 1** | **Configuration of I/O module 3** | |
| | 0 = | I/O module is a dummy module |
| | 1 = | I/O module is not a dummy module |
| **etc.** | **Configuration of I/O module 4 ... 24** | |
| | 0 = | I/O module is a dummy module |
| | 1 = | I/O module is not a dummy module |

| Module register properties | |
|---|---|
| Value after reset | Non-volatile; factory setting: -1 |
| Takes effect | Next time when the controller is launched |

**R 200002024**

## Slave dummy modules

Each bit in this register represents a slave module on the JX2 system bus.

| Meaning of the individual bits | | |
|---|---|---|
| **Bit 0** | **Configuration of slave module 2** | |
| | 0 = | Module is a slave dummy module |
| | 1 = | Module is not a slave dummy module |
| **Bit 1** | **Configuration of slave module 3** | |
| | 0 = | Module is a slave dummy module |
| | 1 = | Module is not a slave dummy module |
| **etc.** | **Configuration of slave module 4 ... 17** | |
| | 0 = | Module is a slave dummy module |
| | 1 = | Module is not a slave dummy module |

| Module register properties | |
|---|---|
| Value after reset | Non-volatile; factory setting: 65,535 |
| Takes effect | Next time when the controller is launched |

**R 200002029**

## Baud rate of JX2 system bus

The user configures the baud rate of the JX2 system bus. A reduced baud rate allows longer lines.

| Values | |
|---|---|
| 7 | 1,000 kBaud |
| 6 | 500 kBaud |
| 5 | 250 kBaud |
| 4 | 125 kBaud |

| Module register properties | |
|---|---|
| Value after reset | Non-volatile; factory setting: 7 |
| Takes effect | Next time when the controller is launched |

**R 200002032**

## ON delay

After the JC-360 has been energized it waits the period given in this register before it starts initializing the JX2 system bus.

| Values | |
|---|---|
| 20 ... 600 | ON delay from 2 s to 60 s |

| Module register properties | |
|---|---|
| Value after reset | Non-volatile,<br>Factory setting: 60 (ON delay = 6 s) |
| Takes effect | Next time when the controller is launched |

**R 200002077**

## Enabling JX2 system bus special functions

The value of this register influences the behavior at initializing the JX2 system bus.

| Meaning of the individual bits | | |
|---|---|---|
| **Bit 3, bit 2** | **Activate CAN-Prim in addition to JX2 system bus** | |
| | 01 = | The CAN-Prim interface and the JX2 system bus are enabled following the next launch of the JX2 system bus. This requires a restart of the controller.<br>This function lets you connect JX2 expansion modules. |
| **Bit 3, bit 2** | **Enable CAN-Prim and CANopen® STX API only.** | |
| | 1x = | At the next restart, the JX2 system bus is not initialized. The CAN-Prim interface can be used.<br>All node IDs can be used **without any** restrictions.<br>The controller does not initialize any JX2 expansion modules on the JX2 system bus. For this reason, JX2 expansion modules **cannot** be connected.<br>The CANopen® STX API can be used. |
| **Bit 6** | **Lets you disable the CANopen® support on the JX2 system bus** | |
| | 1 = | CANopen® support in the JX2 system bus driver is disabled. This requires a restart of the controller.<br>If you intend to connect to the JX2 system bus JX2-I/O and JX2 slave modules along with CANopen® devices, the communication of which you intend to program via CANopen® STX API (mixed operation), you must set bit 6.<br>The JX2 system bus driver supports certain CANopen® devices. If you intend to connect CANopen® devices which are not supported and communication of which you intend to program via CANopen® STX API, you must set this bit to completely disable the support by the JX2 system bus driver.<br>If you intend to program communication with CANopen® devices via CANopen® STX API yourself, only WAGO I/O System 750, JX-SIO, LioN-S modules, etc. can be connected to the JX2 system bus.<br>Bit 6 makes sense only if bit 3 has not been set |

| Module register properties | |
|---|---|
| Value after reset | Non-volatile; factory setting: 0 |
| Takes effect | Next time when the controller is launched |

**Restrictions at mixed operation**

Mixed operation means that the following devices are connected to the JX2 system bus:

- JX2-I/O modules
- JX2 slave modules
- CANopen® devices which are not supported by the JX2 system bus driver and which must be programmed via CANopen® STX API

In this case, the following restrictions apply:

- Only the ranges 50 ... 59 and 70 ... 79 are permitted to be node IDs for the CANopen® devices!
- If the node ID range 50 ... 59 must be applied, only **nine** JX2-I/O modules at the most can still be connected to the JX2 system bus. Otherwise, the CAN IDs used will overlap with the PDO IDs.
  **Important note:** There is no scanning by the controller.
- To be able to connect more than nine JX2-I/O modules, the COB IDs of TX and RX PDOs used by CANopen® devices must be mapped to the ID range 70 ... 79.
  **Important note:** Whether it is possible to make changes to the COB ID of PDOs depends on the corresponding CANopen® device.
- The non-volatile register 200002077 lets you disable the CANopen® support on part of the JX system bus driver. For more information refer to the sample codes below:

```
// JX2 system bus driver not supporting CANopen®
// and CAN-Prim disabled
  Regs[200002077] := 0x40;
// JX2 system bus driver not supporting CANopen®
// and CAN-Prim enabled
  Regs[200002077] := 0x44;
```

  If you intend to program communication with CANopen® devices via CANopen® STX API yourself, only WAGO I/O System 750, JX-SIO, LioN-S modules, etc. can be connected to the JX2 system bus.
- To have the changes accepted, relaunch the controller once.

# Register description of modules connected to the JX2 system bus

**R 200002013**

## Amount of connected I/O modules

The controller JC-360 enters the sum of the following I/O modules into this register:

- JX2-I/O modules
- IP67 module
- CANopen® modules
- I/O dummy modules
- CANopen® dummy modules

| Values | |
|---|---|
| 0 ... 31 | Number of I/O modules |

| Module register properties | |
|---|---|
| Type of access | Read only |
| Value after reset | Amount of connected I/O modules |

**R 200002014**

## Number of connected slave modules

The controller JC-360 enters the sum of the following slave modules into this register:

- JX2 slave modules
- JetMove 1xx, 2xx, and 6xx
- CANopen® slaves
- JX2-Slave dummy modules

| Values | |
|---|---|
| 0 ... 16 | Number of slave modules |

| Module register properties | |
|---|---|
| Type of access | Read only |
| Value after reset | Number of connected slave modules |

**R 200002015** **Index to module array**

This index lets you select the module array entry contained in R 200002016.

| Values | |
|---|---|
| 0 | R 200002016 contains the number of modules connected to the JX2 system bus. |
| 1 ... 51 | R 200002016 contains the module code that has been entered into the module array by the controller. |

**R 200002016** **Module array**

This register value reflects the module code that has been selected in R 200002015 *Index to module array*.

| Module register properties | |
|---|---|
| Type of access | Read only |
| Value after reset | Number of connected I/O and slave modules |

**R 200002070** **Number of connected CANopen® modules**

The controller JC-360 enters into this register the number of CANopen® modules connected to the JX2 system bus.

| Values | |
|---|---|
| 0 ... 10 | Number of connected CANopen® modules |

| Module register properties | |
|---|---|
| Type of access | Read only |

**R 200002071** **Actual I/O sum of modules on the JX2 system bus**

The controller JC-360 calculates the I/O sum of the connected modules and enters it into this register.

| Module register properties | |
|---|---|
| Type of access | Read only |

# Register description - Error logging on the JX2 system bus

**R 200002008** **Error registers - JX2 system bus**

If an error on the JX2 system bus occurs, the controller enters its cause into this register.

| Meaning of the individual bits | | |
|---|---|---|
| **Bit 3** | **I/O module timeout** | |
| | 1 = | At least one I/O module has caused a timeout |
| **Bit 4** | **JX2 slave module timeout** | |
| | 1 = | At least one JX2 slave module has caused a timeout |
| **Bit 9** | **Peripheral fault** | |
| | 1 = | At least one I/O module suffers a peripheral fault, e.g. short-circuit or overload |
| **Bit 12** | **Object length has not been set** | |
| | 1 = | In case of write access to the CANopen® application registers, the object length has not been set |
| **Bit 13** | **Initialization error** | |
| | 1 = | During JX2 system bus initialization an error occurred |
| **Bit 14** | **Timeout of a system register relating to the JX2 system bus** | |
| | 1 = | Timeout during access to a system register of the JX2 system bus |
| **Bit 15** | **SDO abort** | |
| | 1 = | At SDO access, the CANopen® device has reported an SDO abort |

| Module register properties | |
|---|---|
| Type of access | Only value **0** can be entered.<br>Value **0** lets you acknowledge all reported errors<br>Yet, the E LED at the JC-360 does not go out, unless you have then acknowledged bit 2 (JX2-system bus error) of R 200008. |

**R 200002011** **Number of the I/O module where the timeout has occurred**

If during communication with an I/O module a timeout occurs, the controller enters the number of the I/O module into this register. A timeout might occur in the following cases:

- Sending the monitoring signals within certain intervals
- Read/write access to digital I/O data
- Read/write access to module registers of an I/O module

| Values | |
|---|---|
| 2 ... 32 | I/O module numbers in the case of JX2-I/O modules and IP67 modules |
| 70 ... 79 | I/O module number in the case of CANopen® modules |

| Module register properties | |
|---|---|
| Type of access | Only value **0** can be entered<br>Value **0** lets you delete the entered value |

**R 200002012**

### Number of the slave module where the timeout has occurred

If during communication with a slave module a timeout occurs, the controller enters the number of the slave module into this register. A timeout might occur in the following cases:

- Read/write access to module registers of a slave module

| Values | |
|---|---|
| 2 ... 17 | Slave module number |

| Module register properties | |
|---|---|
| Type of access | Only value **0** can be entered<br>Value **0** lets you delete the entered value |

**Register for error counters**

The following registers function as counters of various CAN errors that might occur on the JX2 system bus.

Detailed knowledge of the Controller Area Network (CAN bus) is required.

| Register | Description |
|---|---|
| 200002821 | Write 1 to set the CAN error counters to 0 |
| 200002824 | Counter for stuff errors |
| 200002825 | Counter for CRC errors |
| 200002826 | Counter for formal errors |
| 200002827 | Counter for acknowledge errors |
| 200002828 | Counter for bit errors |

**R 200002039**

## I/O module where a peripheral fault has occurred

If the JC-360 detects a peripheral fault for an I/O module on the JX2 system bus, it sets the corresponding bit in this register.

| **Meaning of the individual bits** | | |
|---|---|---|
| **Bit 0** | **I/O module 2** | |
| | 1 = | I/O module reports a peripheral fault |
| **Bit 1** | **I/O module 3** | |
| | 1 = | I/O module reports a peripheral fault |
| **etc.** | **I/O module 4 ... 24** | |
| | 1 = | I/O module reports a peripheral fault |

| **Module register properties** | |
|---|---|
| Type of access | Only value **0** can be entered<br>Value **0** lets you delete the entered value |

# Register description - Timeout and interval times for modules on the JX2 system bus

**R 200002028**

## Monitoring interval for I/O modules

This register is for setting the monitoring interval at which the JC-360 checks communication with connected I/O modules.

| Values | |
|---|---|
| 1 ... 255 | Cycle time of the monitoring interval in steps of 10 ms |

| Module register properties | |
|---|---|
| Type of access | Read only |
| Value after reset | 20 [corresponds to a monitoring interval of 200 ms] |

**R 200002073**

## Timeout for register access to CANopen® modules

This timeout applies to access to the following registers:

- CANopen® modules: R 200007000 ... R 20007999
- IP67 modules: R 200003000 ... R 20003249

| Values | |
|---|---|
| 1 ... 255 | Timeout in milliseconds |

| Module register properties | |
|---|---|
| Value after reset | 20 [ms] |

**R 200002074**

## CANopen® SYNC Interval

The controller sends a SYNC telegram according to the CANopen® specification to the bus at configurable intervals. The following CANopen® modules need the SYNC telegram:

- Lenze 2175 CANopen®/DeviceNet for connecting a vector 8200
- Lenze CANopen® PT for connecting a vector 8200

| Values | |
|---|---|
| 0 | SYNC telegram disabled |
| 1 ... 255 | SYNC interval in milliseconds |

| Module register properties | |
|---|---|
| Value after reset | No Lenze vector 8200 present: 0 [ms]<br>Lenze vector 8200 present: 100 [ms] |

**R 200002763**

## Timeout for I/O update of I/O modules

The response to I/O updates of I/O modules must be within the configured timeout time:

- JX2-I/O modules

| Values | |
|---|---|
| 1 ... 255 [ms] | Timeout in milliseconds |

| Module register properties | |
|---|---|
| Value after reset | 10 [ms] |

**R 200002764**

## Timeout at register access to I/O modules

The response to register access to I/O modules must be within the configured timeout time:

- JX2-I/O modules

| Values | |
|---|---|
| 1 ... 255 [ms] | Timeout in milliseconds |

| Module register properties | |
|---|---|
| Value after reset | 20 [ms] |

**R 200002765** **Timeout interval for register access to JX2 modules**

The response to register access to JX2-Slave modules must be within the configured timeout time:

- JX2 slave modules
- JetMove 1xx, 2xx, and 6xx

| **Values** | |
|---|---|
| 1 ... 255 [ms] | Timeout in milliseconds |

| **Module register properties** | |
|---|---|
| Value after reset | 20 [ms] |

# Register description - Retry counter for JX2 system bus modules

**R 200002760**

## Maximum number of I/O update retries

During I/O update the controller tries several times to read or write data Not before the maximum number of retries is exceeded, a timeout error is generated

| Values | |
|---|---|
| 1 ... 255 | Maximum number of retries |

| Module register properties | |
|---|---|
| Value after reset | 5 |

**R 200002761**

## Index to I/O timeout monitoring array

This index is used to select the entry of the I/O retry counter array contained in R 200002762.

| Values | |
|---|---|
| 2 ... 24 | JX2-I/O modules, IP67 modules |
| 70 ... 79 | CANopen® modules |

**R 200002762**

## I/O retry counter array

This array contains the sum of all retries occurred during I/O update of the corresponding module.

| Values | |
|---|---|
| 0 ... 255 | Number of retries |

# Register description - Versions of JX2 system bus drivers

**Introduction**

Apart from information on the OS version of the JC-360 there is also additional version information for identifying the JX2 system bus driver.

**R 200002000** **Version of the JX2 system bus interface**

| Module register properties | |
|---|---|
| Type of access | Read only |
| Data type | IP format |

**R 200002072** **Version of the JX2 system bus driver**

| Module register properties | |
|---|---|
| Type of access | Read only |
| Data type | IP format |

**R 200002995** **Version of the boot loader for the JX2 system bus interface**

| Module register properties | |
|---|---|
| Type of access | Read only |
| Data type | IP format |

# 10.12 JX3 system bus

**Introduction**

JX3 modules are directly connected to the JX3 system bus of the controller JC-3xx or bus node JX3-BN-xxx. The JX3 system bus supports different modules. Access to these modules is completely transparent.

**Configuration**

The JX3 system bus needs not be configured. The controller JC-3xx and the bus node JX3-BN-xxx are able to automatically detect and commission connected modules.

**Connectable modules**

- JX3 modules

**Contents**


| Topic | Page |
|---|---|
| Module array and module codes of connected modules | 432 |
| Dummy modules on the JX3 system bus | 434 |
| JX3 system bus - Description of non-volatile registers | 435 |
| Register description - Modules detected on the JX3 system bus | 436 |
| Register description - Error logging on the JX3 system bus | 437 |
| Register description - Timeout intervals on the JX3 system bus | 439 |
| Register description - Versions of JX3 system bus drivers | 440 |

# Module array and module codes of connected modules

**Module array**

A unique module code is assigned to each module that is connected to the JX3 system bus. The controller JC-3xx, and the bus node JX3-BN-xxx enter all modules detected during initialization into the module array.

Modules are entered into the module array in the following order:

- As connected
- JX3-PS1 modules are not entered into the module array.

**Registers - Overview**

| Register | Description |
|---|---|
| **R 100002015** | Index to module array |
| **R 100002016** | Module array<br>R 100002015 = 0: Number of modules<br>R 100002015 = 1: Module code of the first module<br>etc. |

**Reading the module array**

The module array is accessed indirectly.

| Step | Action |
|---|---|
| **1** | Enter the number of the module in the module array into R 100002015 *Index to Module Array*. |
| **2** | Read out the module code from R 100002016 *Module Array*. |

**Module codes of JX3 modules**

| Module code | Module | Description |
|---|---|---|
| 300 | JX3-DI16 | 16 digital inputs |
| 301 | JX3-DIO16 | 16 digital inputs and 8 digital outputs |
| 302 | JX3-DO16 | 16 digital outputs |
| 303 | JX3-AI4 | 4 analog inputs |
| 304 | JX3-AO4 | 4 analog outputs |
| 305 | JX3-MIX1 | Multi-purpose module |
| 307 | JX3-THI2-RTD | 2 inputs for resistance thermometers |
| 308 | JX3-CNT | Universal counter module |
| 310 | JX3-MIX2 | Multi-purpose module |
| 312 | JX3-THI2-TC | 2 inputs for thermocouples |
| 316 | JX3-DMS2 | 2 inputs for strain gages |
| 340 | JX3-AI4-EI | 4 analog inputs with galvanic isolation |
| 341 | JX3-THI2-RTD-EI | 2 inputs for resistance thermometers with galvanic isolation |
| 342 | JX3-THI2-TC-EI | 2 inputs for thermocouples with galvanic isolation |

**Module codes - Dummy modules**

| Module code | Module | Description |
|---|---|---|
| 251 | I/O dummy module | - |

# Dummy modules on the JX3 system bus

**Dummy modules**

A dummy module is a module on the JX3 system bus that actually does not exist. When assigning I/O module numbers, the controller JC-360 and the bus node JX3-BN-xxx treat dummy modules as if they were existing modules.

Dummy modules allow the user to insert a nonexistent module between existing modules.

**Registers - Overview**

The value contained in the register for dummy module configuration is non-volatile. Any changes become effective only after the module has been re-initialized.

| Register | Description |
|---|---|
| **R 100002023** | I/O dummy modules |

**Allowed dummy modules**

- JX3 modules

**Configuring dummy modules**

| Step | Action |
|---|---|
| **1** | Modify the dummy module configuration within the controller via R 100002023. |
| **2** | Switch the controller off. |
| **3** | Then, switch the controller on.<br>**Result:**<br>The controller or bus node has initialized the JX3 system bus taking into account the configured dummy modules. Information from the dummy modules can be retrieved via module array. |

**R 100002023**

**Dummy modules**

| **Module register properties** | |
|---|---|
| Values | 1 ... 65535 (bit-coded)<br>Bit 0 -> module 2<br>Bit 1 -> module 3<br>...<br>Bit 15 -> module 17<br><br>Bit = 1: Module may exist<br>Bit = 0: Dummy module |
| Value after reset | 65535 (all modules may exist) |

# JX3 system bus - Description of non-volatile registers

**Introduction**

Non-volatile registers let you configure the JX3 system bus. Any changes to a non-volatile register become effective only after the JC-360 has been re-booted.

**R 100002023**

## I/O dummy modules

Each bit in this register represents an I/O module on the JX3 system bus.

| Meaning of the individual bits | | |
|---|---|---|
| **Bit 0** | **Configuration of I/O module 2** | |
| | 0 = | I/O module is a dummy module |
| | 1 = | I/O module is not a dummy module |
| **Bit 1** | **Configuration of I/O module 3** | |
| | 0 = | I/O module is a dummy module |
| | 1 = | I/O module is not a dummy module |
| **etc.** | **Configuration of I/O module 4 ... 17** | |
| | 0 = | I/O module is a dummy module |
| | 1 = | I/O module is not a dummy module |

| Module register properties | |
|---|---|
| Value after reset | Non-volatile; factory setting: 65535 |
| Takes effect | Next time when the controller is launched |

**R 100002034**

## Number of retries

This register lets you set the number of retries in accessing the JX3 modules. Before you make changes to this value consult the hotline at Jetter AG.

| Module register properties | |
|---|---|
| Values | 1 ... 5 |
| Value after reset | Non-volatile; factory setting: 1 |
| Takes effect | Next time when the controller is launched |

# Register description - Modules detected on the JX3 system bus

**R 100002013**

## Number of detected I/O modules

The controller JC-3xx and the bus node JX3-BN-xxx enter the sum of the following I/O modules into this register:

- JX3-I/O modules
- I/O dummy modules

| Values | |
|---|---|
| 0 ... 16 | Number of I/O modules |

| Module register properties | |
|---|---|
| Type of access | Read only |
| Value after reset | Amount of connected JX3-I/O modules |

**R 100002015**

## Index to module array

This index lets you select the module array entry contained in R 100002016.

| Values | |
|---|---|
| 0 | R 100002016 contains the number of modules connected to the JX3 system bus. |
| 1 ... 16 | R 100002016 contains the module code that has been entered into the module array by the controller. |

**R 100002016**

## Module array

This register value reflects the module code that has been selected in R 100002015 *Index to module array*.

| Module register properties | |
|---|---|
| Type of access | Read only |
| Value after reset | Amount of connected JX3-I/O modules |

# Register description - Error logging on the JX3 system bus

**R 100000000**

## Bus status

The controller or bus node enters into this register the status of the JX3 system bus.

| Meaning of the individual bits | |
|---|---|
| **Bit 15** | **Bus status** |
| | 1 = Data exchange takes place via JX3 system bus. |

| Module register properties | |
|---|---|
| Type of access | Read access |
| Value after reset | Depending on the initialization state |

**R 100002008**

## JX3 system bus - Error registers

If an error on the JX3 system bus occurs, the controller or the bus node enters its cause into this register.

| Meaning of the individual bits | |
|---|---|
| **Bit 3** | **Error** |
| | 1 = At least one JX3 module has caused an error. |
| **Bit 16** | **Fatal error** |
| | 1 = A fatal, non-recoverable error has occurred on the JX3 system bus. Data exchange has been terminated. |

| Module register properties | |
|---|---|
| Type of access | Only 0 can be entered. |
| Value after reset | Depending on the initialization state |

**R 100002011**

## Number of the I/O module where the error has occurred

If during communication with an I/O module an error occurs, the controller enters the number of the I/O module into this register. An error might occur in the following cases:

- Read/write access to process data of JX3 modules
- Read/write access to module registers of a JX3 module

| Values | |
|---|---|
| 2 ... 17 | Number of the JX3 I/O module |

| Module register properties | |
|---|---|
| Type of access | Only 0 can be entered. |

**R 100002111**

## Register number of the module where the error has occurred

If during communication with an I/O module an error occurs, the controller enters the number of the module register into this register. An error might occur in the following cases:

- Read/write access to module registers of a JX3 module

| Values | |
|---|---|
| -1 ... 9999 | Module register number of the JX3 module |

| Module register properties | |
|---|---|
| Type of access | Only 0 can be entered. |
| Value after reset | -1 |

# Register description - Timeout intervals on the JX3 system bus

**R 100002764**

**Timeout interval for register access to JX3 modules**

The response to register access to JX3 modules must be within the configured timeout interval:

- JX3 modules

| **Values** | |
|---|---|
| 1 ... 255 [ms] | Timeout in milliseconds |

| **Module register properties** | |
|---|---|
| Value after reset | 15 [ms] |

# Register description - Versions of JX3 system bus drivers

**Introduction**

Apart from information on the OS version of the JC-360 there is also additional version information for identifying the JX3 system bus driver.

**R 100002000**

**Version of the JX3 system bus interface**

| Module register properties | |
|---|---|
| Type of access | Read only |
| Data type | IP format |

**R 100002072**

**Version of the JX3 system bus driver**

| Module register properties | |
|---|---|
| Type of access | Read only |
| Data type | IP format |

# 10.13 E-mail

**Introduction**

The user creates template files for e-mails. Into these templates, the controller JC-360 can enter variables for sending, if required. The controller sends e-mails to an e-mail server which will then forward the message.

This chapter gives a description on how to configure the e-mail feature in the JC-360, and on how to create and send e-mails.

**Activating the E-mail feature**

To activate the e-mail feature in the JC-360, the following requirements must be met:

- When ordering the controller option -W was selected.
- A valid e-mail configuration file **/EMAIL/email.ini** exists while the controller is booting.

If both requirements have been met, the corresponding bit in the web status register is set.

**Required programmer's skills**

To be able to use the e-mail feature, the following skills are required:

- Since files are used to configure the e-mail feature, and e-mails as such are based on these files, the user must be familiar with the file system.
- IP networks

**Contents**


| Topic | Page |
|---|---|
| Configuring the E-mail feature | 442 |
| Creating e-mails | 450 |
| Sending an e-mail | 459 |
| Registers | 460 |

# 10.13.1 Configuring the E-mail feature

**Introduction**

This chapter gives a description on how to configure the e-mail feature so as to allow sending e-mails from within the application program.

During the boot process, the JC-360 reads out configuration data from the file **/EMAIL/email.ini**.

**Prerequisites**

For creating the configuration file, the following prerequisites must be fulfilled:

- The IP address of the e-mail server is known.
- If the IP address of the e-mail server is not known, name resolution through a DNS server must be possible - refer to *Using names for IP addresses* (see page 89).
- The log-on and authentication parameters at the e-mail server are known.

To obtain this information contact your network administrator.

**Contents**


| Topic | Page |
|---|---|
| Structure of the configuration file | 443 |
| Section [SMTP] | 444 |
| Section [POP3] | 446 |
| Section [DEFAULT] | 448 |
| Configuration file - Examples | 449 |

## Structure of the configuration file

**Introduction**

The configuration of the e-mail client in the controller is based on the contents of the file **/EMAIL/email.ini**. The JC-360 reads the values during the boot process only.

**Structure of the configuration file**

This configuration file is a text file the entries of which are grouped into several sections.

- These sections are for entering values which are then used by the e-mail client.
- You can insert blank lines as required.
- The following characters precede a comment line: "!", "#" or ";".

**Sections**

The configuration file contains up to three sections. Section [SMTP] is mandatory. The user does not need to create the other sections unless they are actually required.

| Section | Configuration values |
|---|---|
| [SMTP] | ■ IP address and port number of the SMTP server<br>■ Log-on parameters |
| [POP3] | ■ IP address and port number of the POP3 server<br>■ Log-on parameters |
| [DEFAULT] | ■ Name of an e-mail template file containing default values |

# Section [SMTP]

**Introduction**

This section lets you specify the parameters for establishing a connection with the SMTP server.

**Example:**

```
[SMTP]
IP       = 192.168.40.1
PORT     = 25000
HELO     = JetControl_2
USER     = JetControl0815
PASSWORD = MyPassWord
```

**Authentication**

This type of authentication requires the JC-360 to log on at the SMTP server before sending an e-mail. During the logon process USER and PASSWORD must be entered. The JC-360 supports the following authentication procedures:

- LOG-ON
- PLAIN
- CRAM-MD5

**Configuration values**

| **IP** | |
|---|---|
| In the given example | 192.168.40.1 |
| Description | IP address of the SMTP server;<br>can also be specified as name. |
| Allowed values | ■ > 1.0.0.0<br>■ < 223.255.255.255 |
| Illegal values | ■ Network address<br>■ Broadcast address |
| In case of illegal value or missing entry | The e-mail feature is not available |

| **PORT** | |
|---|---|
| In the given example | 25.000 |
| Description | Port number of the SMTP server |
| Allowed values | ■ > 0<br>■ < 65.536 |
| Illegal values | ■ > 65,335 |
| In case of missing entry | 25 |

| **HELO** | |
|---|---|
| In the given example | JetControl_2 |
| Description | Name for logging on at the e-mail server |
| Allowed values | String of 63 characters max. |
| In case of missing entry | When sending the e-mail, the controller uses the entry contained in [FROM]. |

| **USER** | |
|---|---|
| In the given example | JetControl0815 |
| Description | Log-on name for SMTP authentication.<br>If this entry exists, the entry PASSWORD is required, too. |
| Allowed values | String of 63 characters max. |
| In case of missing entry | SMTP authentication will not be carried out. |

| **PASSWORD** | |
|---|---|
| In the given example | MyPassWord |
| Description | Log-on password for SMTP authentication.<br>If this entry exists, the entry USER is required, too. |
| Allowed values | String of 63 characters max. |
| In case of missing entry | SMTP authentication will not be carried out. |

# Section [POP3]

**Introduction**

This section lets you specify the parameters for establishing a connection with the POP3 server.

Only in case the e-mail server requires authentication via POP3-before-SMTP, this section is required.

**Example:**

```
[POP3]
IP       = 192.168.40.1
PORT     = 25100
USER     = JetControl4711
PASSWORD = Pop3PassWord
```

**Authentication**

This type of authentication requires the JC-360 to log on at the POP3 server. During the logon process USER and PASSWORD must be entered. After that, the SMTP server allows e-mails to be sent for a given period of time (usually 10 to 30 minutes).

**Configuration values**

| **IP** | |
|---|---|
| In the given example | 192.168.40.1 |
| Description | IP address of POP3 server;<br>can also be specified as name. |
| Allowed values | ■ > 1.0.0.0<br>■ < 223.255.255.255 |
| Illegal values | ■ Network address<br>■ Broadcast address |
| In case of illegal value or missing entry | POP3 log-on will not be carried out. |

| **PORT** | |
|---|---|
| In the given example | 25.100 |
| Description | Port number of POP3 server |
| Allowed values | ■ > 0<br>■ < 65.536 |
| Illegal values | ■ > 65,335 |
| In case of missing entry | 110 |

| **USER** | |
|---|---|
| In the given example | JetControl4711 |
| Description | Log-on name for POP3 authentication.<br>If this entry exists, the entry PASSWORD is required, too. |
| Allowed values | String of 63 characters max. |
| In case of missing entry | POP3 log-on will not be carried out. |

| **PASSWORD** | |
|---|---|
| In the given example | Pop3PassWord |
| Description | Log-on password for POP3 authentication.<br>If this entry exists, the entry USER is required, too. |
| Allowed values | String of 63 characters max. |
| In case of missing entry | POP3 log-on will not be carried out. |

## Section [DEFAULT]

**Introduction**

In this section, specify the name of an e-mail template file which contains default settings for e-mails. If the respective section is not available in the respective e-mail template, the JC-360 applies these default settings for sending an e-mail message.

**Example**

```
[DEFAULT]
MAILCFG = EmailDefaults.cfg
```

**Related topics**

- **Structure of template file** (see page 452)

# Configuration file - Examples

**Introduction**

This section contains several examples of the e-mail configuration file **/EMAIL/email.ini**.

**Minimum configuration**

If no authentication is required and the default value is assigned to the IP port of the SMTP server, the configuration file must contain only the IP address of the SMTP server.

```
[SMTP]
IP       = 192.168.40.1
```

**Authentication through POP3 Log-on**

In case the e-mail server requires previous log-on through POP3 and an e-mail template containing default setting has been defined:

```
[SMTP]
IP       = 192.168.40.1

[POP3]
IP       = 192.168.40.1
USER     = JetControl4711
PASSWORD = Pop3PassWord

[DEFAULT]
MAILCFG = EmailDefaults.cfg
```

**Authentification through SMTP**

In case the e-mail server requires an encrypted authentication:

```
[SMTP]
IP       = 192.168.40.1
USER     = JetControl0815
PASSWORD = MyPassWord
```

# 10.13.2 Creating e-mails

**Introduction**

This chapter describes how to create an e-mail. Then, the application program sends these e-mails.

For each e-mail the user has to create an e-mail template file.

**Contents**


| Topic | Page |
|---|---|
| Name of the e-mail template file | 451 |
| Structure of the e-mail template file | 452 |
| Inserting real-time controller values | 454 |

# Name of the e-mail template file

**Introduction**

This naming convention must only be kept to, if system function 110 is applied, which should not be used any further, though.

The STX feature `EMailSend()` lets you select any file name and directory, as long as the limitations owed to the file system are kept to.

The name of an e-mail template file consists of a fixed part of the name and a variable part. The variable part of the name allows the application program to choose various e-mails for sending.

**File name**

`email_#.cfg`

| Part of the name | Description |
|---|---|
| email_ | Name prefix which always remains fixed |
| # | Number of the e-mail; value between 0 and 255 |
| .cfg | Fixed file extension |

**Storage location**

E-mail template files must be stored to the same directory on the internal flash disk as is the configuration file.

`/EMAIL`

**Examples**

```
email_0.cfg
email_37.cfg
email_255.cfg
```

# Structure of the e-mail template file

**Introduction**

An e-mail template file is a text file which is divided into sections. For sending the e-mail, the JC-360 compiles the information contained in these sections.

**E-mail template file**

- Sections [FROM] and [TO] are mandatory. This information may be specified either in the e-mail to be sent or in the e-mail template file containing the default settings.
- All parameters in these sections can be tagged with realtime controller values (refer to *Inserting realtime controller values* (see page 208)).

[FROM]
Sender

[TO]
Addressee

[CC]
Additional addressee(s)

[SUBJECT]
Subject

[ATTACHMENT]
Complete path and file name

[MESSAGE]
E-mail message text

**Sections**

| **[FROM]** | |
|---|---|
| Description | E-mail sender |
| Comments | Please check with your IT administrator which information must be entered here. |
| Length | 63 characters |
| Example | `[FROM]`<br>`JetControl@jetter.de` |

| **[TO]** | |
|---|---|
| Description | E-mail addressee |
| Comments | Several addressees are separated by the semicolon “;”. |
| Length | 255 characters |
| Example | `[TO]`<br>`service@mydomain.com` |

| **[CC]** | |
|---|---|
| Description | Additional e-mail addressee(s) |
| Comments | Several addressees are separated by the semicolon “;”. |
| Length | 255 characters |
| Example | `[CC]`<br>`service@mydomain.com;hotline@mydomain.com` |

| **[SUBJECT]** | |
|---|---|
| Description | Subject |
| Length | 255 characters |
| Example | `[SUBJECT]`<br>`Fatal Error` |

| **[ATTACHMENT]** | |
|---|---|
| Description | Complete name of the file to be attached |
| Comments | The attached file must be a text file. |
| Length | 511 characters |
| Example | `[ATTACHMENT]`<br>`/logfiles/error_report.log` |

| **[MESSAGE]** | |
|---|---|
| Description | E-mail message text |
| Comments | Text only message |
| Length | 65,535 characters |
| Example | `[MESSAGE]`<br>`Have a nice day !`<br>`JetControl.` |

# Inserting real-time controller values

**Introduction**

Actual real-time controller values are integrated into parameter entries within the sections via tag functions. This way, the contents respectively states of registers, text registers, inputs, outputs and flags can be displayed.

**Tag delimiters**

All tags start and end with defined strings (delimiters). Between these tag delimiters, the variables are defined.

| Delimiter | String |
|---|---|
| Tag start | <JW:DTAG |
| Tag end | /> |

**Variable definition**

The variable definition in a tag contains attributes which are used to set, for example, how the value of a variable is to be displayed:

| **name** | |
|---|---|
| Function | Variable name |
| Comments | Code letter followed by the variable number |
| Example | `name="R1000023"` |

| **type** | |
|---|---|
| Description | Variable type of notation |
| Example | `type="REAL"` |

| **format** | |
|---|---|
| Description | Representation format |
| Comments | Refer to format definition |
| Example | `format="+0####.###"` |

| **factor** | |
|---|---|
| Description | Factor by which the real-time controller value is multiplied |
| Comments | Multiplication is executed prior to adding the offset |
| Example | `factor="1.5"` |

| **offset** | |
|---|---|
| Description | Value which is added to the real-time controller value |
| Comments | Multiplication by the factor is executed prior to adding the value to the real-time controller value |
| Example | `offset="1000"` |

**Format definition**

You can define the representation of variables by means of their attribute.

- The number of digits/characters used for representing a variable can be defined by the character "#".
- Prefix "0" sets the output of leading zeros. This applies to the register types INT, INTX and REAL.
- Prefix "+" sets the output of a sign. This applies to the register types INT and REAL.
- Prefixing a blank sets the output of a blank. This applies to the register types INT and REAL.

**Registers/text registers**

The variable name begins with a capital "R" followed by the register number.
The following types are possible:

| Type | Notation |
|---|---|
| INT (standard type) | Integer, decimal |
| INTX | Integer, hexadecimal |
| INTB | Integer, binary |
| BOOL | Register content = 0 --> Display: 0<br>Register content != 0 --> Display: 1 |
| REAL | Floating point, decimal |
| STRING | Text register |

**Example:**

```
JW:DTAG name="R1000250" type="REAL" format="+0####.###"
factor="3.25" offset="500" /
```

**Result:**

This instruction causes the contents of register 1000250 to be multiplied by 3.25. Then 500 is added to the product. The result appears in the Web browser with sign and at least five integer positions before the decimal point. Leading zeros are added as appropriate. Furthermore, three decimal positions are added.

**Flags**

The variable name begins with a capital "F" followed by the flag number.
The following types are possible:

| Type | Notation |
|---|---|
| BOOL (standard type) | Flag = 0 --> Display: 0<br>Flag = 1 --> Display: 1 |
| STRING | Flag = 0 --> Display: FALSE<br>Flag = 1 --> Display: TRUE |

**Example:**

```
<JW:DTAG name="F100" type="STRING" format="#" />
```

**Result:**
The state of flag 100 is displayed as string "T" or "F".

**Inputs**

The variable name begins with a capital "I" followed by the input number.
The following types are possible:

| Type | Notation |
|---|---|
| BOOL (standard type) | Input = 0 --> Display: 0<br>Input = 1 --> Display: 1 |
| STRING | Input = 0 --> Display: OFF<br>Input = 1 --> Display: ON |

**Example:**

```
<JW:DTAG name="I100000308" type="STRING" />
```

**Result:**
The state of input 100000308 is displayed as string "ON" or "OFF".

**Outputs**

The variable name begins with a capital "O" followed by the output number.
The following types are possible:

| Type | Notation |
|---|---|
| BOOL (standard type) | Output = 0 --> Display: 0<br>Output = 1 --> Display: 1 |
| STRING | Output = 0 --> Display: OFF<br>Output = 1 --> Display: ON |

**Example:**

```
<JW:DTAG name="O100000308" />
```

**Result:**
The state of output 100000308 is inserted as "1" or "0".

**Access via pointer register**

Access via pointer register is realized by inserting the capital letter "P" in front of the variable name. In each case the value of the variable is displayed whose number corresponds to the content of the register specified in the variable name.

**Examples:**

```
<JW:DTAG name="PR1000300" />
```

Result: The content of the register is displayed whose number is contained in register 1000300.

```
<JW:DTAG name="PF1000300" />
```

Result: The state of the flag is displayed whose number is contained in register 1000300.

```
<JW:DTAG name="PI1000300" />
```

Result: The state of the input is displayed whose number is contained in register 1000300.

```
<JW:DTAG name="PO1000300" />
```

Result: The state of the output is displayed whose number is contained in register 1000300.

**Access via pointer register and offset**

To specify the number of the variable to be displayed, it is also possible to add a constant value or another register content to the pointer register value

**Examples:**

```
<JW:DTAG name="PR1000300 + 100" />
```

**Result:** The content of the register is displayed whose number results from the addition of the content of register 1000300 and value 100.

```
<JW:DTAG name="PR1000300 + R1000100" />
```

**Result:** The content of the register is displayed whose number results from the addition of the content of register 1000300 and the content of register 1000100.

```
<JW:DTAG name="PF1000300 + 100" />
```

**Result:** The state of the flag is displayed whose number results from the addition of the content of register 1000300 and value 100.

```
<JW:DTAG name="PF1000300 + R1000100" />
```

**Result:** The state of the flag is displayed whose number results from the addition of the content of register 1000300 and the content of register 1000100.

```
<JW:DTAG name="PI1000300 + 100" />
```

**Result:** The state of the input is displayed whose number results from the addition of the content of register 1000300 and the value 100.

```
<JW:DTAG name="PI1000300 + R1000100" />
```

**Result:** The state of the input is displayed whose number results from the addition of the content of register 1000300 and the content of register 1000100.

```
<JW:DTAG name="PO1000300 + 100" />
```

**Result:** The state of the output is displayed whose number results from the addition of the content of register 1000300 and the value 100.

```
<JW:DTAG name="PO1000300 + R1000100" />
```

**Result:** The state of the output is displayed whose number results from the addition of the content of register 1000300 and the content of register 1000100.

## 10.13.3 Sending an e-mail

| | |
|---|---|
| **Introduction** | This chapter gives a description on how to send previously created e-mails from within the application program.<br>When sending an e-mail from the application program, the device JC-360 creates the e-mail based on the e-mail template file and inserts variable values if required. |
| **Processing within the application program** | Sending an e-mail may take considerable time. Therefore, other tasks of the application program are processed while an e-mail is being sent. Only a function call via e-mail is not possible. While an e-mail of a task is being sent, all other tasks which invoke the e-mail function are therefore blocked until this operation is completed. |
| **System function 110** | As of JetSym 5.0, system function 110 is outdated. Instead, apply JetSym STX function `EMailSend()`. |
| **JetSym STX function EMailSend()** | The JetSym STX function EMailSend() has been described in detail in the online help of JetSym.<br>Declaration of functions:<br>`Function EmailSend(Const Ref FileName: String): Int;` |

# 10.13.4 Registers

**Introduction**

This chapter gives a description of those registers from which you can query the status of e-mail processing.

**Contents**


| Topic | Page |
|---|---|
| Overview of registers | 461 |
| Registers - Description | 462 |

# Overview of registers

**Introduction**

The device JC-360 makes the registers available from which you can query the status of e-mail processing.

**Register overview**

| Register | Description |
|---|---|
| **202930** | Web status |
| **292932** | IP address of the SMTP server |
| **292933** | IP address of the POP3 server |
| **292934** | Port number of the SMTP server |
| **292935** | Port number of POP3 server |
| **292937** | Status of e-mail processing |
| **292938** | ID of the task that is just sending an e-mail |

## Registers - Description

**R 202930**

**Web status**

The Web status register displays all available functions in bit-coded mode.

| Meaning of the individual bits | |
|---|---|
| **Bit 0** | **FTP server**<br>1 = available |
| **Bit 1** | **HTTP server**<br>1 = available |
| **Bit 2** | **E-mail**<br>1 = available<br>Bit 2 is only set, if configuration file **/EMAIL/email.ini** exists. |
| **Bit 3** | **Data file function**<br>1 = available |
| **Bit 4** | **Modbus/TCP**<br>1 = existing |
| **Bit 5** | **Modbus/TCP**<br>1 = available |
| **Bit 7** | **FTP client**<br>1 = available |

| Module register properties | |
|---|---|
| Type of access | Read |
| Value after reset | Depending on options purchased |

**R 292932**

**IP address of the SMTP server;**

This register lets you read the IP address of the SMTP server in the same format as specified in the file **/EMAIL/email.ini**.

| Module register properties | |
|---|---|
| Type of access | Read |
| Value after reset | Depending on configuration |
| Takes effect | Once R 202930.2 = 1 |

**R 292933** **IP address of POP3 server**

This register lets you read the IP address of the POP3 server in the same format as specified in the file **/EMAIL/email.ini**.

| **Module register properties** | |
|---|---|
| Type of access | Read |
| Value after reset | Depending on configuration |
| Takes effect | Once R 202930.2 = 1 |

**R 292934** **Port number of the SMTP server**

This register lets you read the port number of the SMTP server in the same format as specified in the file **/EMAIL/email.ini**.

| **Module register properties** | |
|---|---|
| Type of access | Read |
| Value after reset | Depending on configuration |
| Takes effect | Once R 202930.2 = 1 |

**R 292935** **Port number of POP3 server**

This register lets you read the port number of the POP3 server in the same format as specified in the file **/EMAIL/email.ini**.

| **Module register properties** | |
|---|---|
| Type of access | Read |
| Value after reset | Depending on configuration |
| Takes effect | Once R 202930.2 = 1 |

**R 292937**

## Status of e-mail processing

This register lets you track the e-mail status.

| Module register properties | | |
|---|---|---|
| Values | 0 | No e-mail is being sent |
| | 1 | Parameters are being handed over to the e-mail client of the JC-360. |
| | 2 | E-mail is being compiled and connection with the server is being established. |
| | 3 | E-mail has been sent to the server. |
| Type of access | Read | |

**R 292938**

## Task ID (e-mail)

The ID of the task that is just sending an e-mail can be seen from this register.

| Module register properties | | |
|---|---|---|
| Values | 0 ... 99 | Task ID |
| | 255 | There is no task sending an e-mail. |
| Value after reset | 255 | |
| Type of access | Read | |

# 10.14 Sorting data

| | |
|---|---|
| **Introduction** | This chapter describes system function 50. This system function is used to trigger the sorting algorithm provided by the operating system. |
| **Application** | For sorting data in controller registers by their value.<br><br>The sort algorithm is provided by the operating system of the controller. The data to be sorted are indirectly addressed through a descriptor using parameter 1. |
| **System function 50** | As of JetSym 5.0, system function 50 is outdated. Instead, apply JetSym STX function `QSort()`. |
| **JetSym STX function QSort()** | The JetSym STX function QSort() has been described in detail in the online help of JetSym.<br>Declaration of functions:<br>`Function QSort(DataPtr: Int, ElementCnt: Int, ElementSize: Int, SortOffset: Int, SortType: STXBASETYPE, SortMode: QSORTMODE): Int;` |

# 10.15 Modbus/TCP

**Introduction**

This chapter describes the functions of the Modbus/TCP server and client integrated into JC-360.

**Enabling the Modbus/TCP feature**

In case of JC-340, you must order the controller plus option -M. In any other controller of this series, the feature Modbus/TCP remains activated.

If this requirement has been met, bits 4 and 5 in the Web status register 202930 are set.

**Required programmer's skills**

To be able to use the functions described in this chapter, the following skills are required:

- The user must be familiar with Modbus/TCP and the supported commands.
- IP networks

**Contents**


| Topic | Page |
|---|---|
| Modbus/TCP server | 467 |
| Modbus/TCP client | 473 |
| Modbus/TCP client with STX variables | 475 |

# 10.15.1 Modbus/TCP server

**Introduction**

If a valid license is available, if the Modbus/TCP feature has been enabled and the Modbus/TCP server has been launched, an external client can access registers, flags, inputs and outputs.

This chapter covers the addressing process and describes the commands supported by the Modbus/TCP server.

**Number of possible connections**

Four connections may be opened at the same time.

**Restriction**

Modbus/TCP only supports transmission of registers with a width of 16 bits. From this follows, that only the 16 least significant bits are transmitted when 32-bit registers are sent.

When assigning incoming register values to the internal 32-bit registers no sign extension will be carried out.

**Contents**


| Topic | Page |
|---|---|
| Addressing | 468 |
| Supported commands - Class 0 | 470 |
| Supported commands - Class 1 | 471 |
| Supported commands - Class 2 | 472 |

# Addressing

**Introduction**

The addresses which have been received via Modbus/TCP can be modified locally in the Modbus-TCP server. For this purpose, three registers have been provided. The basic addresses for accessing registers, inputs and outputs are entered into these registers. Then, the address contained in the Modbus/TCP frame specifies the address with reference to the basic address.

**R 272702**

**Register offset**

The basic address for accessing registers via Modbus/TCP is entered into R 272702.

| **Register properties** | |
|---|---|
| Value after reset | 1000000 |

**R 272704**

**Input offset**

The basic address for accessing inputs via Modbus/TCP is entered into R 272704.

| **Register properties** | |
|---|---|
| Value after reset | 100000000 |

**R 272705**

**Output offset**

The basic address for accessing outputs via Modbus/TCP is entered into R 272705.

| **Register properties** | |
|---|---|
| Value after reset | 100000000 |

**Example 1**

The Modbus/TCP server on the JetControl receives from a Modbus/TCP client the command **read multiple registers** starting from register number 100. The number of registers to be read is 5. Register 272702 *Register Offset* contains the value 1000000.

Hence, registers 1000100 through 1000104 will be read.

**Example 2**

The Modbus/TCP server on the JetControl receives from a Modbus/TCP client the command **read input discretes** specifying input number 210 and the instruction to read this input. Register 272704 *Input Offset* contains the value 100000000.

Hence, input 100000210 of a peripheral module JX3-DI16 will be read.

| | |
|---|---|
| **Example 3** | The Modbus/TCP server on the JetControl receives from a Modbus/TCP client the command **write coils** specifying output number 205 and the instruction to set this output. Register 272705 *Output Offset* contains value 100000000.<br>Hence, output 100000205 of a peripheral module JX3-DO16 will be set. |

## Supported commands - Class 0

| | |
|---|---|
| **fc 3** | **read multiple registers** |
| | Reading register sets<br><br>The starting register number within JC-360 is calculated as follows: Register number specified in the command plus the content of R 272702 *Register Offset*. |
| **fc 16** | **write multiple registers** |
| | Writing register sets<br><br>The starting register number within JC-360 is calculated as follows: Register number specified in the command plus the content of R 272702 *Register Offset*. |

## Supported commands - Class 1

| | |
|---|---|
| **fc 1** | **read coils** |
| | Reading outputs<br><br>The output number within the JC-360 is calculated as follows: Output number specified in the command plus the content of register 272705 *Output Offset*. |
| **fc 2** | **read input discretes** |
| | Reading inputs<br><br>The input number within JC-360 is calculated as follows: Input number specified in the command plus the content of register 272704 *Input Offset*. |
| **fc 4** | **read input registers** |
| | Reading inputs blockwise in 16-bit words.<br><br>The starting register number within JC-360 is calculated as follows: Register number specified in the command plus the content of R 272702 *Register Offset*. |
| **fc 5** | **write coil** |
| | Enabling/disabling an individual output<br><br>The output number within the JC-360 is calculated as follows: Output number specified in the command plus the content of register 272705 *Output Offset*. |
| **fc 6** | **write single register** |
| | Entering values into the 16 least significant bits of a register<br><br>The starting register number within JC-360 is calculated as follows: Register number specified in the command plus the content of R 272702 *Register Offset*. |

## Supported commands - Class 2

| | |
|---|---|
| **fc 15** | **force multiple coils** |
| | Enabling/disabling several outputs |
| | The output number within the JC-360 is calculated as follows: Output number specified in the command plus the content of register 272705 *Output Offset*. |
| **fc 23** | **read/write registers** |
| | Reading/writing registers simultaneously |
| | The starting register number within the JC-360 is calculated as follows: Register number specified in the command plus the content of R 272702 *Register Offset*. |

## 10.15.2 Modbus/TCP client

**Introduction**

The Modbus/TCP client included in JC-360 supports only Class 0 Conformance.

In this class, commands for reading and writing multiple registers are used. Up to 125 registers with a width of 16 bits can be transmitted in one frame.

As protocol ID "0" is used. Assignment of sent and received frames is carried out using the transaction ID.

This chapter describes how to carry out noncyclical or cyclical transmission to a Modbus/TCP server using system functions.

**Number of possible connections**

Connections to eleven different Modbus/TCP servers may be opened at the same time.

**Noncyclical data transmission**

System functions 65 and 67 *reading registers*, as well as 66 and 68 *writing registers* are used to establish a noncyclical transmission channel to a Modbus/TCP server.

These system functions establish a connection to the specified Modbus/TCP server, transmit the desired data and clear down the connection.

If RemoteScan has already established a connection for cyclical data transmission, this connection will be used. Setting up and clearing down the connection is, therefore, not required.

**Cyclical data transmission**

Cyclical data transmission is made through the configurable function `RemoteScan`. The inputs and outputs 20001 through 36000 that are combined in the 16-bit registers 278000 through 278999 are cyclically transmitted from and to the Modbus/TCP servers.

Only one connection is established to each Modbus/TCP server (IP address and port) irrespective of the number of communication units which have been configured on this server.

If several communication units are configured on one Modbus/TCP server, accesses are serialized since servers often do not support **command pipelining**. If several servers have been configured, communication is carried out in parallel.

**Combined inputs and outputs**

| Register | Inputs and outputs |
|---|---|
| 278000 | 20001 ... 20016 |
| 278001 | 20017 ... 20032 |
| 278002 | 20033 ... 20048 |
| ... | ... |
| 278999 | 35985 ... 36000 |

These registers and the inputs and outputs mapped to them are merely storage cells within the RAM. There is no direct mapping to the hardware. Therefore, it is not defined whether inputs or outputs are mapped to a register. Assignment is made not until configuration in the communication units takes place.

**Unit ID**

The instruction header of a Modbus/TCP telegram contains a *Unit ID*. The Unit ID is not evaluated by Modbus/TCP devices, as they can be addressed without ambiguity by their IP address. Therefore, in the case of system functions 65, 66 and 80 always value "1" is sent.

Converters from Modbus/TCP to Modbus RTU use the *Unit ID* for addressing the Modbus RTU servers. Therefore, the corresponding system functions for reading and writing registers (system functions 67 and 68), as well as for initializing RemoteScan (system function 85) have been provided. These system functions can be used to set the Unit ID.

**Restriction**

Modbus/TCP only supports transmission of registers with a width of 16 bits. From this follows, that only the 16 least significant bits are transmitted when 32-bit registers are sent.

When assigning incoming register values to the internal 32-bit registers no sign extension will be carried out.

**Outdated system functions**

As of JetSym 5.0, the system functions are outdated. Instead, apply the corresponding JetSym STX functions.

**JetSym STX functions**

This is a comparison between the system functions and the corresponding JetSym STX functions.

| System function | Corresponding JetSym STX function |
|---|---|
| 60 | Function ModbusCRCgen(FramePtr: Int, Length: Int): Int; |
| 61 | Function ModbusCRCcheck(FramePtr: Int, Length: Int): Int; |
| 65/67 | Function ModbusReadReg(Const Ref MbParam: MODBUS_PARAM): Int; |
| 66/68 | Function ModbusWriteReg(Const Ref MbParam: MODBUS_PARAM): Int; |
| 80/85 | Function RemoteScanConfig(Protocol: RSCAN_PROTOCOL, Elements: Int, Const Ref Configuration: RSCAN_DSCR): Int; |
| 81 | Function RemoteScanStart(Protocol: Int): Int; |
| 82 | Function RemoteScanStop(Protocol: Int): Int; |

## 10.15.3 Modbus/TCP client with STX variables

**Introduction**

The Modbus/TCP client included in JC-360 supports only Class 0 Conformance.

In this class, commands for reading and writing multiple registers are used. One frame transmits up to 125 registers of 16 bits width.

As protocol ID, "0" is used. Assignment of transmitted and received frames is carried out using the transaction ID.

This chapter describes how to carry out noncyclical or cyclical transmission to a Modbus/TCP server using STX functions.

**Number of possible connections**

Connections to eleven different Modbus/TCP servers may be opened at the same time.

**Noncyclical data transmission**

Functions `ModbusReadReg()` and `ModbusWriteReg()` are used to establish a noncyclical transmission channel to a Modbus/TCP server.

These functions copy data between registers of a Modbus/TCP server and STX variables. They establish a connection to the specified Modbus/TCP server, transmit the desired data and clear down the connection again.

If RemoteScan has already established a connection for cyclical data transmission, this connection will be used. Setting up and clearing down the connection is, therefore, not required.

**Cyclical data transmission**

Cyclical data transmission is made through the configurable function `RemoteScanConfig()`. The data are cyclically transmitted from and to the Modbus/TCP servers by means of STX variables.

Only one connection is established to each Modbus/TCP server (IP address and port) irrespective of the number of communication units which have been configured on this server.

If several communication units are configured on one Modbus/TCP server, accesses are serialized since servers often do not support **command pipelining**. If several servers have been configured, communication is carried out in parallel.

**Unit ID**

Converters from Modbus/TCP to Modbus RTU use the *Unit ID* for addressing the Modbus RTU servers. For this reason, the Unit ID can be set.

**JetSym STX functions**

The JetSym STX functions have been described in detail in the online help of JetSym.

| System function | Corresponding JetSym STX function |
|---|---|
| 60 | Function ModbusCRCgen(FramePtr: Int, Length: Int): Int; |
| 61 | Function ModbusCRCcheck(FramePtr: Int, Length: Int): Int; |
| 65/67 | Function ModbusReadReg(Const Ref MbParam: MODBUS_PARAM): Int; |
| 66/68 | Function ModbusWriteReg(Const Ref MbParam: MODBUS_PARAM): Int; |
| 80/85 | Function RemoteScanConfig(Protocol: RSCAN_PROTOCOL, Elements: Int, Const Ref Configuration: RSCAN_DSCR): Int; |
| 81 | Function RemoteScanStart(Protocol: Int): Int; |
| 82 | Function RemoteScanStop(Protocol: Int): Int; |

# 10.16 User-programmable serial interface

**Introduction**

This chapter describes how to address the serial interface of the controller from within the application program to allow sending and receiving characters.

**Applications**

The user-programmable serial interface lets you connect devices which use communication protocols that are not supported by the OS of the controller, Fields of application, for example, are:

- Scales
- Scanners
- Display elements
- Frequency inverters
- Temperature controllers
- etc.

**Required programmer's skills**

This chapter addresses programmers of application programs with experience in data transfer via asynchronuous serial interfaces. Expertise in the following areas is prerequisite:

- Wiring of serial interfaces
- Communication parameters (baud rate, parity, etc.)
- Transmit and receive buffers
- etc.

**Contents**


| Topic | Page |
|---|---|
| Interface | 478 |
| Functioning principle of the user-programmable serial interface | 482 |
| Registers | 486 |
| Programming | 495 |

# 10.16.1 Interface

**Introduction**

This chapter covers the connection to a user-programmable serial interface of the JC-360.

**Contents**


| Topic | Page |
|---|---|
| Serial interface port X11 | 479 |

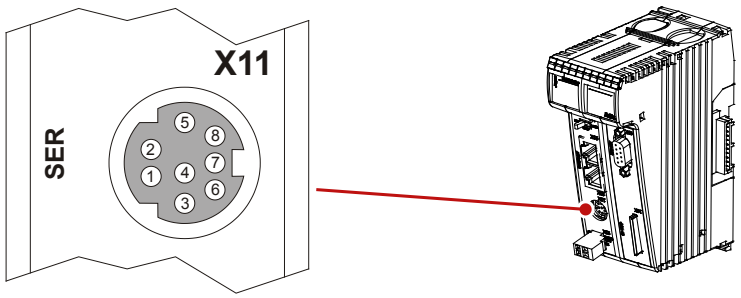
# Serial interface port X11

**Devices to connect with this port**

Port X11 lets you connect the following devices:

- A PC
- An HMI by Jetter AG
- Any device

**Pin assignment of port X11**




| Pin | Signal | Description |
|---|---|---|
| **1** | RDA | RS-422; receive data inverted |
| **2** | GND | Reference potential |
| **3** | RDB | RS-422; receive data not inverted |
| **4** | RxD | RS-232; receive data |
| **5** | SDB | RS-422; transmit data not inverted<br>RS-485; transmit/receive data not inverted |
| **6** | DC 24 V | HMI supply voltage |
| **7** | SDA | RS-422; transmit data inverted<br>RS-485; transmit/receive data inverted |
| **8** | TxD | RS-232; transmit data |

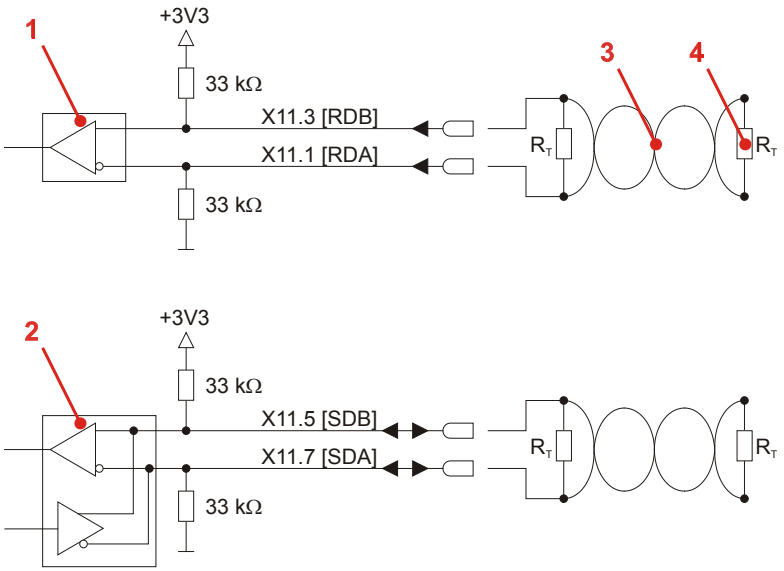
**Restrictions**

Irrespective of the fact that various hardware drivers have been implemented, only one hardware interface is available.

This means:

While, for example, communication via RS-422 is taking place, simultaneous and independent communication via RS-232 is not possible.

### Block diagram




| Number | Part | Function in the case of RS-422 | Function in the case of RS-485 |
|---|---|---|---|
| **1** | Receiver | Receives data | Unused |
| **2** | Receiver/transmitter | Transmits data | Receives and transmits data |
| **3** | Serial line | Twisted line of the serial interface | |
| **4** | $R_T$ | Terminating resistor | |

### Terminating resistor

Connect a terminating resistor to both serial lines in the following cases:

- Long lines
- High baud rates

Select a terminating resistor which corresponds to the impedance of the line used.

**Technical specifications**

| Parameter | Description |
|---|---|
| Type of terminal | MiniDIN, shielded |
| Number of pins | 8 |
| Electrical isolation | None |
| Number of interfaces | 1 serial interface |
| Interface standards | RS-232/RS-422/RS-485-2 |
| Baud rates | 2,400 ... 115,200 baud |
| Bits per character | 5, 6, 7, 8 |
| Number of stop bits | 1, 2 |
| Parity | Even, odd, none, 1, 0 |

**Cables for port X11**

For connecting devices to port X11 you can order the following cables:

| Item no. | Item | Description |
|---|---|---|
| 60867209 | KAY_0576-0050 | JetControl to modem with 9-pin Sub-D, length 0.5 m |
| 60868359 | Cable assy # 196 2.5M | JetControl to PC with 9-pin Sub-D, length 2.5 m |
| 60860013 | Cable assy # 196 5M | JetControl to PC with 9-pin Sub-D, length 5 m |
| 60868956 | Cable assy # 196 8M | JetControl to PC with 9-pin Sub-D, length 8 m |
| 60860011 | Cable assy # 192 2.5M | JetControl to HMI with 15-pin Sub-D, length 2.5 m |
| 60860012 | Cable assy # 193 5M | JetControl to HMI with 15-pin Sub-D, length 5 m |
| 60872142 | Cable assy # 192 10M | JetControl to HMI with 15-pin Sub-D, length 10 m |
| 60872884 | Cable assy # 192 15M | JetControl to HMI with 15-pin Sub-D, length 15 m |
| 60864359 | KAY_0386-0250 | JetControl to LCD 60 with 15-pin Sub-D, length 2.5 m |
| 60864360 | KAY_0386-0500 | JetControl to LCD 60 with 15-pin Sub-D, length 5 m |
| 60864897 | KAY_0533-0025 | JetControl to LCD 52/54 with 15-pin Sub-D, length 0.25 m |
| 60864257 | Cable assy # 197 5M | JetControl to JetView 200/300 with 9-pin Sub-D, length 5 m |
| 60871930 | Cable assy # 197 12M | JetControl to JetView 200/300 with 9-pin Sub-D, length 12 m |

# 10.16.2 Functioning principle of the user-programmable serial interface

**Introduction**

This chapter describes the functioning principle of the user-programmable serial interface.

**Restrictions**

When using the user-programmable serial interface the following restrictions apply:

- Irrespective of the fact that various hardware drivers have been implemented, only one hardware interface is available.
  This means: While, for example, communication via RS-422 is taking place, simultaneous and independent communication via RS-232 is not possible.
- The controller does not execute the pcomX protocol any more.
  This means: This protocol can no longer be used to communicate, for example, with JetSym, JetViewSoft or HMIs via this protocol.

**Contents**


| Topic | Page |
|---|---|
| Functioning principle | 483 |

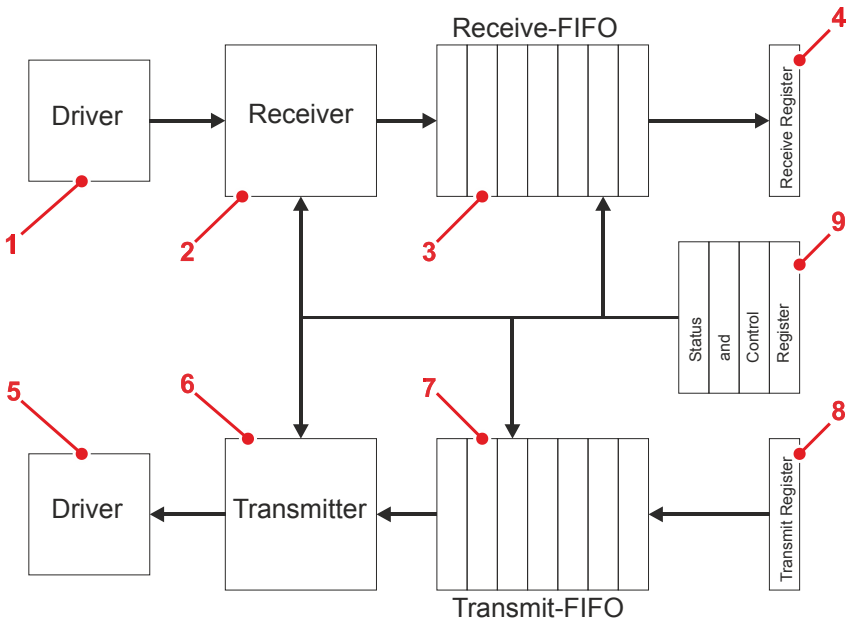
# Functioning principle

**Introduction**

The OS of the JC-360 provides for the user-programmable serial interface a receiving buffer and a transmit buffer. They can be used to adjust the transfer rate between application program and serial interface.

**Block diagram**

The following illustration shows the block diagram of the user-programmable serial interface:




**Elements of the interface**

The user-programmable serial interface consists of the following elements:

| Number | Part | Function |
|---|---|---|
| **1** | Interface driver | Converts signals of different interface standards (RS-232, RS-422, RS-485) into internal signal levels |
| **2** | Addressee | Performs serial/parallel conversion |
| **3** | Receiving buffer | Buffer for received characters |
| **4** | Receive register | Read access to this register reads the received characters in the receive buffer (3). |
| **5** | Interface driver | Converts internal signal levels into signals of different interface standards (RS-232, RS-422, RS-485) |
| **6** | Transmitter | Performs parallel/serial conversion |
| **7** | Transmit buffer | Buffer for characters to be sent |
| **8** | Transmit register | Write access to this register causes the characters to be entered into the transmit buffer (7) and to be sent by the transmitter (6). |

| Number | Part | Function |
|---|---|---|
| **9** | Status and control register | Query of filling levels and error states of buffers; setting of transmission parameters |

**Receiving a character**

A character is received as described below:

| Step | Description |
|---|---|
| **1** | The interface driver converts signals "on the line" into internal signal levels and forwards them to the receiver. |
| **2** | The receiver performs serial/parallel conversion of this character and checks the set communication parameters. |
| **3** | The receiver enters the character into the receive buffer if there is any place left. Otherwise, the character is discarded and buffer overflow is signaled. |
| **4** | Via receive register the character can be read out of the receive buffer. |

**Sending a character**

A character is transmitted as described below:

| Step | Description |
|---|---|
| **1** | Via transmit register the character is entered into the transmit buffer if there is any place left. Otherwise the character is discarded. |
| **2** | Once the transmitter has sent a character, it reads the next character from the transmit buffer. |
| **3** | The transmitter performs parallel/serial conversion and sends this character to the interface driver using the set communication parameters. |
| **4** | The interface driver converts internal signal levels into the various interface standards. |

**Error detection**

When receiving characters, the following errors are detected by the controller and displayed in the register *Error state*:

| Error | Description | Effect |
|---|---|---|
| Framing error | The format of the received character does not match the set parameters. | The erroneous character(s) is (are) stored in the receive buffer and error bit *Framing error* is set. The error counter is incremented. |
| Parity error | The parity bit of the received character is not correct. | The erroneous character is stored in the receive buffer and error bit *Parity error* is set. The error counter is incremented. |
| Buffer overflow | A character is received, although the receive buffer is full. | The character is discarded and error bit *Overflow* is set. The error counter is incremented. |

**Troubleshooting**

As error bits cannot be assigned to individual characters in the receive buffer, all characters should be removed from the receive buffer and discarded when an error bit is set.

Possible causes of error and troubleshooting:

| Error | Possible cause | Troubleshooting |
|---|---|---|
| Framing error | Jammed data transmission caused by EMC problems, defective cables or connectors | ■ Check the wiring and connectors.<br>■ Use shielded cables.<br>■ Do not lay cables near sources of interference. |
| | Incorrectly set communication parameters (baud rate, number of stop bits, etc.) | ■ Make sure the set communication parameters are consistent with the settings of the connected device. |
| Parity error | Jammed data transmission caused by EMC problems, defective cables or connectors | ■ Check the wiring and connectors.<br>■ Use shielded cables.<br>■ Do not lay cables near sources of interference. |
| | Incorrectly set parity | ■ Make sure the parity setting is consistent with the setting of the connected device. |
| Buffer overflow | The external device sends characters at too high a rate and the application program is not able to read them out of the receive buffer in due time. | ■ Program a software handshake.<br>■ Set a lower baud rate.<br>■ Make sure that characters are read out from the receive buffer faster. To achieve this the program code has to be optimized. |

# 10.16.3 Registers

**Introduction**

This chapter describes the registers associated with the user-programmable serial interface. These registers are used for the following tasks:

- Parameterizing the interface
- Sending characters
- Receiving characters

**Contents**


| Topic | Page |
|---|---|
| Register numbers | 487 |
| Registers - Description | 488 |

# Register numbers

**Introduction**

The registers of each interface are combined into one register block. The basic register number of this block is dependent on the controller.

**Register numbers**

| Controller | Basic register number | Register numbers |
|---|---|---|
| JC-360 | 103000 | 103000 ... 103019 |

**Determining register numbers**

In this chapter, only the last two figures of a register number are specified, e.g. MR 14. To calculate the complete register number, add the basic register number of the corresponding device to the respective device, e.g. 103000.

**Registers - Overview**

| Register | Description |
|---|---|
| **MR 0** | Error state |
| **MR 1** | Protocol |
| **MR 2** | Baud rate |
| **MR 3** | Number of data bits per character |
| **MR 4** | Number of stop bits |
| **MR 5** | Parity |
| **MR 6** | Interface standard |
| **MR 10** | Transmit buffer |
| **MR 11** | Transmit buffer filling level |
| **MR 12** | Receive buffer (without deleting characters on reading) |
| **MR 13** | Receive buffer (with deleting characters on reading) |
| **MR 14** | Receive buffer filling level |
| **MR 15** | Receive buffer, 16-bit, little endian |
| **MR 16** | Receive buffer, 16-bit, big endian |
| **MR 17** | Receive buffer, 32-bit, little endian |
| **MR 18** | Receive buffer, 32-bit, big endian |
| **MR 19** | Error counter |

# Registers - Description

**Introduction**

When entering values into control registers MR 1 through MR 6, the entire interface is re-initialized and the transmit and receive buffers are cleared.

**MR 0**

## Error state

This register displays errors which have been detected on receiving a character as bit-coded value.

| Meaning of the individual bits | | |
|---|---|---|
| **Bit 12** | **Buffer overflow** | |
| | 1 = | Although the receiving buffer is full, one or more characters have been received |
| **Bit 13** | **Parity error** | |
| | 1 = | The parity bit of the received character is not correct. |
| **Bit 14** | **Framing error** | |
| | 1 = | The format of the received character does not match the set parameters. |

| Module register properties | |
|---|---|
| Type of access | Read/write (clearing) |

**MR 1**

## Protocol

This register lets you set the protocol which is supported by the OS of the controller. That is, this register is for defining how the interface is used.

| Module register properties | | |
|---|---|---|
| Values | 1 | System logger |
| | 2 | User-programmable Interface |
| | 3 | PcomX |
| Value after reset | 3 | |

**MR 2**

## Baud rate

This register lets you set the baud rate.

| Module register properties | | |
|---|---|---|
| Values | 2,400 ... 115,200 | |
| Value after reset | 9,600 | |

**MR 3**

## Number of data bits per character

This register lets you set the number of data bits per character.

| Module register properties | | |
|---|---|---|
| Values | 5, 6, 7, 8 | |
| Value after reset | 8 | |

**MR 4**

## Stop bits

This register lets you set the number of stop bits per character.

| Module register properties | | |
|---|---|---|
| Values | 1 | 1 stop bit |
| | 2 | 1.5 stop bits if MR 3 = 5 |
| | | 2 stop bits if MR 3 = 6, 7, 8 |
| Value after reset | 1 | |

**MR 5**

## Parity

This register lets you set the parity of a character.

| Module register properties | | |
|---|---|---|
| Values | 0 | None (no parity) |
| | 1 | Odd parity |
| | 2 | Even parity |
| | 3 | 1 (mark) |
| | 4 | 0 (space) |
| Value after reset | 2 | |

**MR 6**

## Interface standard

This register lets you set the hardware interface which is used to receive and transmit characters.

| Module register properties | | |
|---|---|---|
| Values | 0 | RS-232 |
| | 1 | RS-422 |
| | 2 | Reserved |
| | 3 | RS-485, 2-wire |
| Value after reset | 1 | |

**MR 10**

## Transmit buffer

The character that has to be sent must be entered into this register.

- If the transmit buffer is able to accomodate the character, it is entered into this buffer. This character will be sent once all previously entered characters have been sent.
- Prior to sending characters from the application program, it must be checked whether the transmit buffer is able to accommodate characters. This can be checked by reading out MR 11.
- The transmit buffer functions according to the FIFO principle. The first character entered is sent first.

| Module register properties | | |
|---|---|---|
| Values | 0 ... 31 | 5 bits per character |
| | 0 ... 63 | 6 bits per character |
| | 0 ... 127 | 7 bits per character |
| | 0 ... 255 | 8 bits per character |
| Type of access | Read | Character written last |
| | Write | Sending a character |

**MR 11**

## Sending buffer filling level

This register shows how many characters the transmit buffer accomodates. There is space for 32,768 characters max. within the buffer.

| Module register properties | |
|---|---|
| Values | 0 ... 32,768 |

**MR 12**

## Receive buffer, 8 bits (without deleting the character on reading)

This register shows the "oldest" character stored in the receive buffer. On reading, this character will not be removed from the buffer.

| Module register properties | | |
|---|---|---|
| Values | 0 ... 31 | 5 bits per character |
| | 0 ... 63 | 6 bits per character |
| | 0 ... 127 | 7 bits per character |
| | 0 ... 255 | 8 bits per character |
| Type of access | Read | Oldest character in buffer |
| Takes effect | if MR 14 > 0 | |

**MR 13**

## Receive buffer, 8 bits (with deleting the character on reading)

This register shows the "oldest" character stored in the receive buffer. This character is removed from the buffer. Thus, the character received next can be read out during the next read access.

| Module register properties | | |
|---|---|---|
| Values | 0 ... 31 | 5 bits per character |
| | 0 ... 63 | 6 bits per character |
| | 0 ... 127 | 7 bits per character |
| | 0 ... 255 | 8 bits per character |
| Type of access | Read | Oldest character in buffer |
| Takes effect | if MR 14 > 0 | |

**MR 14**

## Receiving buffer filling level

This register shows how many characters the receive buffer accomodates. Each read access to MR 13 decrements this register by 1.

| Module register properties | |
|---|---|
| Values | 0 ... 32,768 |

**MR 15** **Receive buffer, 16-bit, little endian**

Read access to this register removes 2 characters from the receive buffer and returns them as 16-bit value.

**Assignment:**

| Character | Bits in register |
|---|---|
| First | Bit 0 ... 7 |
| Second | Bit 8 ... 15 |

| **Module register properties** | | |
|---|---|---|
| Values | 0 ... 65,535 | |
| Type of access | Read | Removes 2 characters from the buffer |
| Takes effect | if MR 14 > 1 | |

**MR 16** **Receive buffer, 16-bit, big endian**

Read access to this register removes 2 characters from the receive buffer and returns them as 16-bit value.

Assignment:

| Character | Bits in register |
|---|---|
| First | Bit 8 ... 15 |
| Second | Bit 0 ... 7 |

| **Module register properties** | | |
|---|---|---|
| Values | 0 ... 65,535 | |
| Type of access | Read | Removes 2 characters from the buffer |
| Takes effect | if MR 14 > 1 | |

**MR 17**

## Receive buffer, 32-bit, little endian

Read access to this register removes 4 characters from the receive buffer and returns them as 32-bit value.

Assignment:

| Character | Bits in register |
|---|---|
| First | Bit 0 ... 7 |
| Second | Bit 8 ... 15 |
| Third | Bit 16 ... 23 |
| Fourth | Bit 24 ... 31 |

| Module register properties | | |
|---|---|---|
| Values | -2,147,483,648 ... 2,147,483,647 | |
| Type of access | Read | Removes 4 characters from the buffer |
| Takes effect | if MR 14 > 3 | |

**MR 18**

## Receive buffer, 32-bit, big endian

Read access to this register removes 4 characters from the receive buffer and returns them as 32-bit value.

Assignment:

| Character | Bits in register |
|---|---|
| First | Bit 24 ... 31 |
| Second | Bit 16 ... 23 |
| Third | Bit 8 ... 15 |
| Fourth | Bit 0 ... 7 |

| Module register properties | | |
|---|---|---|
| Values | -2,147,483,648 ... 2,147,483,647 | |
| Type of access | Read | Removes 4 characters from the buffer |
| Takes effect | if MR 14 > 3 | |

**MR 19**

## Error counter

This register shows the number of detected errors.

| Module register properties | |
|---|---|
| Values | 0 ... 2,147,483,647 |
| Type of access | Read/write (clearing) |

# 10.16.4 Programming

**Introduction**

This chapter describes how to configure the serial interface of the controller for use as user-programmable serial interface and how to send receive characters via this interface.

**Contents**


| Topic | Page |
|---|---|
| Configuring the interface | 496 |
| Sending characters | 497 |
| Sending texts | 498 |
| Sending values | 499 |
| Receiving characters | 500 |
| Receiving values | 501 |

## Configuring the interface

**Introduction**

Module registers MR 1 through MR 6 are used to configure the user-programmable serial interface.

**Prerequisites**

This guide proceeds from the assumption that wiring between controller and remote device is according to the standard of the selected interface.

**Configuring the interface**

To configure the user-programmable serial interface proceed as follows:

| Step | Action |
|---|---|
| **1** | Enter value 1 into MR 2. |
| **2** | Enter the desired communication parameters into MR 2 through MR 6. |

**Result:** The serial interface is set as user-programmable interface. Both the transmit buffer and receive buffer are cleared.

# Sending characters

**Introduction**

A character is sent by entering it into the register *Transmit buffer*.

**Prerequisites**

This guide proceeds from the assumption that the user-programmable serial interface has been configured.

**Sending characters**

To send characters via user-programmable serial interface proceed as follows:

| Step | Action |
|---|---|
| **1** | Check the transmit buffer filling level, whether there is enough space in the transmit buffer. |
| **2** | If there is no space in the transmit buffer, wait, until there is enough space. |
| **3** | Enter the character to be sent into register *Transmit buffer*. |

**Result:** The character is written into the transmit buffer and will be sent from there.

# Sending texts

**Introduction**

An easy way to send texts via the user-programmable serial interface is redirecting the instructions `DisplayText()` and `DisplayText2()` to **Device 9**.

**Prerequisites**

This guide proceeds from the assumption that the following conditions are met:

- The user-programmable serial interface is configured.
- The user is familiar with the options of the instructions `DisplayText()` and `DisplayText2()` (refer to the online help which comes with JetSym).

**Restrictions**

When redirecting the instructions `DisplayText()` and `DipslayText2()` to the user-programmable serial interface the following restrictions apply:

- The cursor position will not be evaluated.
- The characters for "Delete Screen" and "Delete to End of Line" are of no special significance and will be output without any changes.

**Sending Texts**

To send texts via user-programmable serial interface proceed as follows:

| Step | Action |
|---|---|
| 1 | Use the instruction `DisplayText()` or `DisplayText2()`. |
| 2 | Specify **Device 9**. |

**Result:** The task waits at this instruction until all characters have been entered into the transmit buffer.

# Sending values

**Introduction**

The instruction `DisplayValue()` allows redirection of values to **Device 9**. This way, values can easily be sent via user-programmable serial interface.

**Prerequisites**

This guide proceeds from the assumption that the following conditions are met:

- The user-programmable serial interface is configured.
- The user is familiar with the options of the instruction `DisplayValue()` (refer to the online help which comes with JetSym).

**Restrictions**

When redirecting instruction `DisplayValue()` to the user-programmable serial interface the following restriction applies:

- The cursor position will not be evaluated.

**Sending Values**

To send values via user-programmable serial interface proceed as follows:

| Step | Action |
|---|---|
| **1** | The special registers for formatting the display, which are used in connection with the instruction `DisplayValue()`, have to be set to the desired values. |
| **2** | Use the instruction `DisplayValue()`. |
| **3** | Specify **Device 9**. |

**Result:** The task waits at this instruction until all characters have been entered into the transmit buffer.

## Receiving characters

**Introduction**

A character is received by reading characters from register *Receiving buffer*.

**Prerequisites**

This guide proceeds from the assumption that the user-programmable serial interface has been configured.

**Receiving characters**

To receive characters via user-programmable serial interface proceed as follows:

| Step | Action |
|---|---|
| **1** | Check the filling level of the receiving buffer to make sure that it contains at least 1 character. |
| **2** | Read the character from the register *Receiving buffer*. |

**Result:** The character is taken from the receiving buffer.

# Receiving values

**Introduction**

Values are received by reading characters from registers MR 15 through MR 18 *Receiving buffer registers*.

**Prerequisites**

This guide proceeds from the assumption that the user-programmable serial interface has been configured.

**Receiving values**

To receive values via user-programmable serial interface proceed as follows:

| Step | Action |
|---|---|
| **1** | Check the filling level of the receive buffer to make sure that it contains at least 2 or 4 characters. |
| **2** | Read the values from *Receiving buffer* registers MR 15 through MR 18. |

**Result:** The characters are read from the receiving buffer.

# 10.17 User-programmable IP interface

**The user-programmable IP interface**

The user-programmable IP interface allows to send or receive any data via Ethernet interface on the JC-360 using TCP/IP or UDP/IP. When using this feature, data processing is completely carried out by the application program.

**Applications**

The user-programmable IP interface allows the programmer to carry out data exchange via Ethernet connections which do not use standard protocols, such as FTP, HTTP, JetIP or Modbus/TCP. The following applications are possible:

- Server
- Client
- TCP/IP
- UDP/IP

**Required programmer's skills**

To be able to program user-programmable IP interfaces the following knowledge of data exchange via IP networks is required:

- IP addressing (e.g. IP address, port number, subnet mask)
- TCP (e.g. connection establishment/termination, data stream, data backup)
- UDP (e.g. datagram)

**Restrictions**

For communication via user-programmable IP interface, the programmer must not use any ports which are already used by the operating system of the controller. Therefore, do not use the following ports:

| Protocol | Port number | Default value | User |
|---|---|---|---|
| TCP | Depending on the FTP client | 20 | FTP server (data) |
| TCP | 21 | | FTP server (controller) |
| TCP | 23 | | System logger |
| TCP | 80 | | HTTP server |
| TCP | From the file /EMAIL/email.ini | 25, 110 | E-mail client |
| TCP | 502 | | Modbus/TCP server |
| TCP, UDP | 1024 - 2047 | | Various |
| TCP, UDP | IP configuration | 50000, 50001 | JetIP |
| TCP | IP configuration | 52000 | Debug server |

**Contents**


| Topic | Page |
|---|---|
| Programming | 504 |
| Registers | 516 |

# 10.17.1 Programming

**Introduction**

The user-programmable IP interface is used to carry out data exchange between application program and network client via TCP/IP or UDP/IP connections. For this purpose, function calls are used. These function calls are included in the programming language of the JC-360. To program this feature proceed as follows:

| Step | Action |
|---|---|
| **1** | Initializing the user-programmable IP interface |
| **2** | Open connections |
| **3** | Transfer data |
| **4** | Terminate the connections |

**Technical data**

Technical data of the user-programmable IP interface:

| Feature | Description |
|---|---|
| Number of connections | 20 |
| Maximum data size | 4,000 byte |

**Restrictions**

While the controller JC-360 is processing one of the functions of the user-programmable IP interface, tasks having called the functions should not be stopped through `TaskBreak` or restarted through `TaskRestart`.

Failure to do so could result in the following errors:

- Connections do not open
- Data loss during sending or receiving
- Connections remain open unintentionally
- Connections are closed unintentionally

**Contents**


| Topic | Page |
|---|---|
| Initializing the user-programmable IP interface | 505 |
| Establishing a connection | 506 |
| Sending data | 510 |
| Receiving data | 512 |
| Terminating a connection | 515 |

# Initializing the user-programmable IP interface

**Introduction**

This function must be initialized each time the application program is launched.

**Function declaration**

```
Function ConnectionInitialize():Int;
```

**Return value**

The following return value is possible:

| Return value | |
|---|---|
| 0 | Always |

**How to use this function**

The function is used and its return value assigned to a variable for further utilization in the following way:

```
Result := ConnectionInitialize();
```

**Operating principle**

The device JC-360 processes this function in the following steps:

| Step | Description |
|---|---|
| **1** | The device JC-360 closes all open connections of the user-programmable IP interface. |
| **2** | The device JC-360 initializes all OS-internal data structures of the user-programmable IP interface. |

**Related topics**

- **Establishing a connection** (see page 506)
- **Terminating a connection** (see page 515)
- **Sending data** (see page 510)
- **Receiving data** (see page 512)

# Establishing a connection

**Introduction**

Before data can be sent or received, a connection has to be established. Here, the following criteria have to be discerned:

- Which transaction log (TCP or UDP) has to be used?
- Is it a client or a server that has to be installed?

**Function declaration**

```
Function ConnectionCreate(ClientServerType:Int,
                          IPType:Int,
                          IPAddr:Int,
                          IPPort:Int,
                          Timeout:Int):Int;
```

**Function parameters**

Description of the function parameters:

| Parameter | Value | Comment |
|---|---|---|
| `ClientServerType` | Client = 1 = CONNTYPE_CLIENT<br>Server = 2 = CONNTYPE_SERVER | |
| `IPType` | UDP/IP = 1 = IPTYPE_UDP<br>TCP/IP = 2 = IPTYPE_TCP | |
| `IPAddr` | Valid IP address | Required only for TCP/IP client |
| `IPPort` | Valid IP port number | Will be ignored for UDP/IP client |
| `Timeout` | 0 ... 1,073,741,824 [ms] | 0 = infinitely |

**Return value**

If the return value was positive, the connection could be established. If the returned value was negative, an error occurred and the connection could not be established.

| Return value | |
|---|---|
| > 0 | A positive return value must be stored in a variable. It must be made available as a handle at activating the functions `Send data`, `Receive data`, and `Terminate connection`. |
| -1 | Error during connection set-up |
| -2 | Internal error |
| -3 | Invalid parameter |
| -8 | Timeout |

**Using this function with a TCP/IP client**

If a client is to establish a TCP/IP connection to a server, you can invoke the function and assign the return value of a variable for further evaluation as follows:

```
Result := ConnectionCreate(CONNTYPE_CLIENT,
                           IPTYPE_TCP,
                           IP#192.168.75.123,
                           46000,
                           T#10s);
```

**Functioning principle with a TCP/IP client**

The task stops at the program line until the connection is established or the specified timeout has elapsed. This function is processed in the following steps:

| Step | Description | |
|---|---|---|
| 1 | The device JC-360 tries to establish a TCP/IP connection via port 46000 to the network client with IP address 192.168.75.123. | |
| 2 | **If ...** | **... then ...** |
| | the network client has accepted the connection, | the function is terminated and a positive value is returned as handle for further access to the connection. |
| | the connection could not be established and the timeout of 10 seconds has not elapsed yet, | step 1 is carried out. |
| | an error has occurred or the timeout has elapsed, | the function is terminated and a negative value is returned. |

**Using this function with a TCP/IP server**

If a server is to establish a TCP/IP connection to a client, you can invoke the function and assign the return value of a variable for further evaluation as follows:

```
Result := ConnectionCreate(CONNTYPE_SERVER,
                           IPTYPE_TCP,
                           0,
                           46000,
                           T#100s);
```

**Functioning principle with a TCP/IP server**

The task stops at the program line until the connection is established or the specified timeout has elapsed. This function is processed in the following steps:

| Step | Description | |
|---|---|---|
| 1 | The device JC-360 sets up TCP/IP port 46000 for receiving connection requests. | |
| 2 | **If ...** | **... then ...** |
| | the network client has established a connection, | no further connection requests to this port are accepted, the function is terminated and a positive value is returned as handle for further access to the connection. |
| | the connection could not be established and the timeout of 100 seconds has not elapsed yet, | the system waits for a connection to be established. |
| | an error has occurred or the timeout has elapsed, | the function is terminated and a negative value is returned. |

**Using this function with a UDP/IP client**

If a client is to establish a UDP/IP connection to a server, you can invoke the function and assign the return value of a variable for further evaluation as follows:

```
Result := ConnectionCreate(CONNTYPE_CLIENT,
                           IPTYPE_UDP,
                           0,
                           0,
                           0);
```

**Functioning principle with a UDP/IP client**

UDP is a connectionless communication mode. For this reason, the device JC-360 opens only one communication channel for sending data to a network client. This function is processed in the following steps:

| Step | Description | |
|---|---|---|
| 1 | The device JC-360 sets up a UDP/IP communication channel for sending data. | |
| 2 | **If ...** | **... then ...** |
| | no error has occurred, | the function is terminated and a positive value is returned as handle for further access to the connection. |
| | an error has occurred, | the function is terminated and a negative value is returned. |

**Using this function with a UDP/IP server**

If a server is to establish a UDP/IP connection to a server, you can invoke the function and assign the return value of a variable for further evaluation as follows:

```
Result := ConnectionCreate(CONNTYPE_SERVER,
                           IPTYPE_UDP,
                           0,
                           46000,
                           0);
```

**Functioning principle with a UDP/IP server**

UDP is a connectionless communication mode. For this reason, the device JC-360 opens only one communication channel for receiving data from a network client. This function is processed in the following steps:

| Step | Description | |
|---|---|---|
| 1 | The device JC-360 sets up a UDP/IP communication channel at port 46000 for receiving data. | |
| 2 | **If ...** | **... then ...** |
| | no error has occurred, | the function is terminated and a positive value is returned as handle for further access to the connection. |
| | an error has occurred, | the function is terminated and a negative value is returned. |

**Related topics**

- **Terminating a connection** (see page 515)
- **Sending data** (see page 510)
- **Receiving data** (see page 512)
- **Initializing the user-programmable IP interface** (see page 505)

# Sending data

**Introduction**

Data can be sent via a previously established TCP/IP connection or via a UDP/IP connection of a client.

Via UDP/IP connection of a server data cannot be sent, but only received.

**Function declaration**

```
Function ConnectionSendData(IPConnection:Int,
                            IPAddr:Int,
                            IPPort:Int,
                            Const Ref SendData,
                            DataLen:Int):Int;
```

**Function parameters**

Description of the function parameters:

| Parameter | Value | Comment |
|---|---|---|
| `IPConnection` | Handle | Return value of the function `ConnectionCreate()` |
| `IPAddr` | Valid IP address | Required only for UDP/IP client |
| `IPPort` | Valid IP port number | Required only for UDP/IP client |
| `SendData` | Address of the data block to be sent | |
| | 1 ... 4,000 | Data block length in bytes |

**Return value**

The following return values are possible:

| Return value | |
|---|---|
| 0 | Data have been sent successfully. |
| -1 | Error when sending, e.g. connection interrupted. |
| -3 | Invalid handle, e.g. sending via a UDP/IP server. |

**Using this function with a TCP/IP connection**

If data are to be sent via a TCP/IP connection, you can invoke the function and assign the return value of a variable for further evaluation as follows:

```
Result := ConnectionSendData(hConnection,
                             0,
                             0,
                             SendBuffer,
                             SendLen);
```

**Functioning principle with a TCP/IP connection**

When using TCP/IP, data are sent via a previously opened connection. Therefore, specification of the IP address and IP port number is not required anymore and can be ignored in the function.

In the following cases, the task is not processed further after issuing the function call:

- The data have been sent and their reception confirmed.
- An error has occurred.

**Using this function with a UDP/IP client**

If, with a client, data are to be sent via a UDP/IP connection, you can invoke the function and assign the return value of a variable for further evaluation as follows:

```
Result := ConnectionData(hConnection,
                         IP#192.168.75.123,
                         46000,
                         SendBuffer,
                         SendLen);
```

**Functioning principle with a UDP/IP client**

With UDP/IP there is no connection between two given network clients. Therefore, with each function call data can be sent to another client or another port. The task will pause at the function call until the data are sent.

You will not get any acknowledgment of the remote network client having received the data.

**Related topics**

- **Initializing the user-programmable IP interface** (see page 505)
- **Establishing a connection** (see page 506)
- **Terminating a connection** (see page 515)
- **Receiving data** (see page 512)

# Receiving data

**Introduction**

Data can be sent via a previously established TCP/IP connection or via a UDP/IP connection of a server.

Via UDP/IP connection of a client data cannot be received, but only sent.

**Function declaration**

```
Function ConnectionReceiveData(IPConnection:Int,
                               Ref IPAddr:Int,
                               Ref IPPort:Int,
                               Ref ReceiveData,
                               DataLen:Int,
                               Timeout:Int):Int;
```

**Function parameters**

Description of the function parameters:

| Parameter | Value | Comment |
|---|---|---|
| `IPConnection` | Handle | Return value of the function `ConnectionCreate()` |
| `IPAddr` | Address of a variable for saving the IP address of the sender | Required only for UDP/IP server |
| `IPPort` | Address of a variable for saving the IP port number of the sender | Required only for UDP/IP server |
| `ReceiveData` | Address of the data block to be received | |
| `DataLen` | 1 ... 4,000 | Maximum data block length in bytes |
| `Timeout` | 0 ... 1,073,741,824 [ms] | 0 = infinitely |

**Return value**

The following return values are possible:

| Return value | |
|---|---|
| > 0 | Number of received data bytes |
| -1 | Error when receiving data, e.g. connection interrupted. |
| -3 | Invalid handle, e.g. receiving data via a UDP/IP client. |
| -8 | Timeout |

**Using this function with a TCP/IP connection**

If data are to be received via a TCP/IP connection, you can invoke the function and assign the return value of a variable for further evaluation as follows:

```
Result := ConnectionReceiveData(hConnection,
                                Dummy,
                                Dummy,
                                ReceiveBuffer,
                                sizeof(ReceiveBuffer),
                                T#10s);
```

**Functioning principle with a TCP/IP connection**

When using TCP/IP, data are sent via a previously opened connection. Therefore, specification of the IP address and IP port number is not required any more and can be ignored in the function.

In the following cases, the task is not processed further after issuing the function call:

- The data have been received
- An error has occurred

In case of a TCP/IP connection, data are transmitted as data stream.

The device JC-360 processes this function in the following steps:

| Step | Description | |
|---|---|---|
| 1 | The device JC-360 waits until data have been received, but no longer than the specified timeout. | |
| 2 | **If ...** | **... then ...** |
| | the timeout has elapsed or the connection has been terminated, | the function is exited and an error message is issued. |
| | data have been received, | they are copied to the receiving buffer given along with the data (but not exceeding the amount given along with the data). Then, the function continues with stage 3. |
| 3 | **If ...** | **... then ...** |
| | more data have been received than could have been copied into the receiving buffer, | these are buffered by the JC-360 to be fetched by further function calls. |
| 4 | The function is exited and the number of data, which have been copied into the receiving buffer, is returned. | |

**Using this function with a UDP/IP server**

If, with a server, data are to be received via a UDP/IP connection, you can invoke the function and assign the return value of a variable for further evaluation as follows:

```
Result := ConnectionReceiveData(hConnection,
                                IPAddr,
                                IPPort,
                                ReceiveBuffer,
                                sizeof(ReceiveBuffer),
                                T#10s);
```

**Functioning principle with a UDP/IP server**

In the following situations, the task is not processed further after issuing the function call:

- All data have been received.
- An error has occurred.

In case of a UDP/IP connection, data are transmitted as datagram.
The controller processes this function in the following steps:

| Step | Description | |
|---|---|---|
| 1 | The device JC-360 waits until all data of a datagram have been received, but no longer than the specified timeout. | |
| 2 | **If ...** | **... then ...** |
| | the timeout has elapsed or the connection has been terminated, | the function is exited and an error message is issued. |
| | data have been received, | they are copied to the receiving buffer given along with the data (but not exceeding the amount given along with the data). Then, the function continues with stage 3. |
| 3 | **If ...** | **... then ...** |
| | ... more data have been received than could be copied into the receiving buffer - that is, if the sent datagram is too large, | ... these data are discarded. |
| 4 | The sender's IP address and IP port number are transferred into the variables which are given along with the data. | |
| 5 | The function is exited and the number of data, which have been copied into the receiving buffer, is returned. | |

**Related topics**

- **Initializing the user-programmable IP interface** (see page 505)
- **Establishing a connection** (see page 506)
- **Terminating a connection** (see page 515)
- **Sending data** (see page 510)

# Terminating a connection

**Introduction**

Clear all connections which are no longer required as the number of concurrently opened connections is limited.

**Function declaration**

```
Function ConnectionDelete(IPConnection:Int):Int;
```

**Function parameters**

Description of the function parameters:

| Parameter | Value | Comment |
|---|---|---|
| `IPConnection` | Handle | Return value of the function `ConnectionCreate()` |

**Return value**

The following return values are possible:

| Return value | |
|---|---|
| 0 | Connection terminated and deleted |
| -1 | Invalid handle |

**How to use this function**

This way, you can invoke the function and assign its return value to a variable for further utilization:

```
Result := ConnectionDelete(hConnection);
```

**Related topics**

- **Establishing a connection** (see page 506)
- **Sending data** (see page 510)
- **Receiving data** (see page 512)
- **Initializing the user-programmable IP interface** (see page 505)

# 10.17.2 Registers

**Introduction**

This chapter describes the registers of the JC-360 which contain the current connection list of the user-programmable IP interface. These registers can be used for debugging or diagnostic purposes. However, they can't be used for other functions such as establishing or terminating a connection.

**Contents**


| Topic | Page |
|---|---|
| Register numbers | 517 |
| Register description | 518 |

# Register numbers

**Introduction**

Data of one connection each are displayed within the registers of a coherent register block. The basic register number of this block is dependent on the controller.

**Register numbers**

| Device | Basic register number | Register numbers |
|---|---|---|
| JC-360 | 350000 | 350000 ... 350007 |

**Determining the register number**

In this chapter only the last figure of a register number is specified, for example MR 1. To determine the complete module register number, add to this figure the basic register number of the corresponding device, for example 350000.

**Register overview**

| Register | Description |
|---|---|
| **MR 0** | Selecting a connection |
| **MR 1** | Type of connection |
| **MR 2** | Transport protocol |
| **MR 3** | IP address |
| **MR 4** | IP port number |
| **MR 5** | State |
| **MR 6** | Number of sent bytes |
| **MR 7** | Number of received bytes |

## Register description

**Introduction**

The operating system manages the established connections in a list. Module register MR 0 *Selection of a connection* is used to copy connection details into other registers of a register block.

**MR 0**

**Selecting a connection**

Connections are selected by writing values to this register. Read access to this register is used to display whether the following registers contain connection details.

| **Module register properties** | | |
|---|---|---|
| Reading values | 0 | Connection exists |
| | -1 | Connection does not exist |

| **Module register properties** | | |
|---|---|---|
| Writing values | 0 | Address the first connection in the list |
| | > 0 | Address the next connection in the list |
| | < 0 | Address the previous connection in the list |

**MR 1**

**Type of connection**

The value in this register shows whether the connection is a client or a server connection.

| **Module register properties** | | |
|---|---|---|
| Values | 1 | Client |
| | 2 | Server |

**MR 2**

**Transport protocol**

The value in this register shows whether TCP or UDP is used as transport protocol.

| **Module register properties** | | |
|---|---|---|
| Values | 1 | UDP |
| | 2 | TCP |

**MR 3** **IP address**

The value in this register shows the configured IP address.

| Module register properties | |
|---|---|
| Values | 0,0,0,0 ... 255,255,255,255 |

**MR 4** **IP port number**

The value in this register shows the configured IP port number.

| Module register properties | |
|---|---|
| Values | 0 ... 65,535 |

**MR 5** **State**

The value in this register shows status the connection is currently in.

| Module register properties | | |
|---|---|---|
| Values | 0 | Connection terminated |
| | 1 | Connection is being established |
| | 2 | Connection is established |
| | 3 | TCP/IP server: Waiting for connection request from client |
| | 4 | Internal usage |

**MR 6** **Number of sent bytes**

The value in this register shows the number of data bytes sent via the given connection. Since this is a signed 32-bit register and the sent bytes are added each time, the number range may be exceeded from the positive maximum value to the negative maximum value.

| Module register properties | |
|---|---|
| Values | -2,147,483,648 ... 2,147,483,647 |

**MR 7**

## Number of received bytes

The value in this register shows the number of data bytes received via the given connection. Since this is a signed 32-bit register and the received bytes are added each time, the number range may be exceeded from the positive maximum value to the negative maximum value.

| Module register properties | |
|---|---|
| Values | -2,147,483,648 ... 2,147,483,647 |

# 10.18 CANopen® STX API

**Introduction**

This chapter describes the STX functions of the CANopen® STX API.

**The CANopen® standard**

CANopen® is an open standard for networking and communication in the automation sector.

The CANopen® protocol has been further developed by the CiA e.V. (CAN in Automation) and works on the physical layer with CAN Highspeed in accordance with ISO 11898.

**Documentation**

The CANopen® specifications can be obtained from the **CiA e.V. http://www.can-cia.org** homepage. The key specification documents are:

- CiA DS 301 - This document is also known as the communication profile and describes the fundamental services and protocols used under CANopen®.
- CiA DS 302 - Framework for programmable devices (CANopen® Manager, SDO Manager)
- CiA DR 303 - Information on cables and connectors
- CiA DS 4xx - These documents describe the behavior of a number of device classes in, what are known as, device profiles.

**Application**

These STX functions are used in communication between the JC-360 and other CANopen® nodes.

**Terms and abbreviations**

In this chapter, the following terms and abbreviations are used:

| Term | Description |
|---|---|
| Node ID | Node identification number of the device:<br>This ID lets you address the device. |
| NMT | Network management |
| ro | Read only access |
| rw | Read/write access |

**R 200002077**

## Enabling JX2 system bus special functions

The value of this register influences the behavior at initializing the JX2 system bus.

| Meaning of the individual bits | | |
|---|---|---|
| **Bit 3, bit 2** | **Activate CAN-Prim in addition to JX2 system bus** | |
| | 01 = | The CAN-Prim interface and the JX2 system bus are enabled following the next launch of the JX2 system bus. This requires a restart of the controller.<br>This function lets you connect JX2 expansion modules. |
| **Bit 3, bit 2** | **Enable CAN-Prim and CANopen® STX API only.** | |
| | 1x = | At the next restart, the JX2 system bus is not initialized. The CAN-Prim interface can be used.<br>All node IDs can be used **without any** restrictions.<br>The controller does not initialize any JX2 expansion modules on the JX2 system bus. For this reason, JX2 expansion modules **cannot** be connected.<br>The CANopen® STX API can be used. |
| **Bit 6** | **Lets you disable the CANopen® support on the JX2 system bus** | |
| | 1 = | CANopen® support in the JX2 system bus driver is disabled. This requires a restart of the controller.<br>If you intend to connect to the JX2 system bus JX2-I/O and JX2 slave modules along with CANopen® devices, the communication of which you intend to program via CANopen® STX API (mixed operation), you must set bit 6.<br>The JX2 system bus driver supports certain CANopen® devices. If you intend to connect CANopen® devices which are not supported and communication of which you intend to program via CANopen® STX API, you must set this bit to completely disable the support by the JX2 system bus driver.<br>If you intend to program communication with CANopen® devices via CANopen® STX API yourself, only WAGO I/O System 750, JX-SIO, LioN-S modules, etc. can be connected to the JX2 system bus.<br>Bit 6 makes sense only if bit 3 has not been set |

| Module register properties | |
|---|---|
| Value after reset | Non-volatile; factory setting: 0 |
| Takes effect | Next time when the controller is launched |

**Only CANopen® devices are connected to the JX2 system bus**

If on the JX2 system bus **only** CANopen® devices are connected and CANopen® STX API is used, the JX2 system bus can be completely disabled.

The steps to be taken for this are described below:

| Step | Action |
|---|---|
| **1** | Set bit 3 in R 200002077.<br>`Regs[200002077] := 0x08;` |
| **2** | To have the changes accepted, relaunch the controller once. |
| ⇨ | Now, all node IDs can be used by CANopen® **without** any restrictions. |

**Restrictions at mixed operation**

Mixed operation means that the following devices are connected to the JX2 system bus:

- JX2-I/O modules
- JX2 slave modules
- CANopen® devices which are not supported by the JX2 system bus driver and which must be programmed via CANopen® STX API

In this case, the following restrictions apply:

- Only the ranges 50 ... 59 and 70 ... 79 are permitted to be node IDs for the CANopen® devices!
- If the node ID range 50 ... 59 must be applied, only **nine** JX2-I/O modules at the most can still be connected to the JX2 system bus. Otherwise, the CAN IDs used will overlap with the PDO IDs.

  **Important note:** There is no scanning by the controller.
- To be able to connect more than nine JX2-I/O modules, the COB IDs of TX and RX PDOs used by CANopen® devices must be mapped to the ID range 70 ... 79.

  **Important note:** Whether it is possible to make changes to the COB ID of PDOs depends on the corresponding CANopen® device.
- The non-volatile register 200002077 lets you disable the CANopen® support on part of the JX system bus driver. For more information refer to the sample codes below:

```
// JX2 system bus driver not supporting CANopen®
// and CAN-Prim disabled
  Regs[200002077] := 0x40;
// JX2 system bus driver not supporting CANopen®
// and CAN-Prim enabled
  Regs[200002077] := 0x44;
```

  If you intend to program communication with CANopen® devices via CANopen® STX API yourself, only WAGO I/O System 750, JX-SIO, LioN-S modules, etc. can be connected to the JX2 system bus.
- To have the changes accepted, relaunch the controller once.

**Contents**


| Topic | Page |
|---|---|
| STX function: CanOpenInit() | 525 |
| STX function: CanOpenSetCommand() | 527 |
| STX function: CanOpenUploadSDO() | 529 |
| STX function: CanOpenDownloadSDO() | 532 |
| STX function: CanOpenAddPDORx() | 535 |
| STX function: CanOpenAddPDOTx() | 540 |
| Heartbeat monitoring | 544 |
| CANopen® object dictionary | 548 |

# STX function: CanOpenInit()

**Introduction**

The function `CanOpenInit()` lets you initialize one of the CAN busses. The device then automatically sends the heartbeat message every second with the following communication object identifier (COB-ID): Node ID + 0x700.

**Function declaration**

```
Function CanOpenInit(
    CANNo:Int,
    NodeID:Int,
    const ref SWVersion:String,
) :Int;
```

**Function parameters**

The function `CanOpenInit()` has got the following parameters.

| Parameter | Description | Value |
|---|---|---|
| CANNo | CAN bus number | 0 ... CANMAX |
| NodeID | Node ID of the given device | 1 ... 127 |
| SWVersion | Reference to own software version<br>This software version is entered into the index 0x100A in the object directory. | String up to 255 characters |

**Return value**

This function transfers the following return values to the higher-level program.

| Return value | |
|---|---|
| 0 | OK |
| -1 | Error when checking parameters |
| -3 | Initialization has not worked |
| -4 | The JX2 system bus driver is activated |

**CANNo parameter**

This parameter specifies the number of the CAN interface. CANNo = 0 is assigned to the first interface. The number of CAN interfaces depends on the device. For information on the maximum number of CAN interfaces (CANMAX) refer to the chapters *Technical Specifications* and *Quick Reference* in the corresponding manual.

**How to use this function**

This function lets you initialize CAN bus 0. The device has node ID 20 (0x14).

```
Result := CanOpenInit(0, 20, 'Version: 01.00.0.00');
```

**Operating principle**

During initialization, the device processes the following process steps:

| Step | Description |
|---|---|
| **1** | First, the bootup message is sent as a heartbeat message. |
| **2** | As soon as the device goes into **pre-operational** status, it sends the heartbeat message **pre-operational**. |

**Access to the object directory**

If the device is in **pre-operational** state, it lets you access the object directory using SDO.

**NMT messages**

After initialization, NMT messages can be sent and received. The own heartbeat status can be changed with the function `CanOpenSetCommand`.

**Related topics**

- **STX function CanOpenSetCommand** (see page 527)

# STX function: CanOpenSetCommand()

**Introduction**

The function `CanOpenSetCommand()` lets you change the heartbeat status of the device itself and of all other devices (NMT slaves) on the CAN bus.

**Function declaration**

```
Function CanOpenSetCommand(
    CANNo:Int,
    iType:Int,
    Value:Int,
) :Int;
```

**Function parameters**

The function `CanOpenSetCommand()` has got the following parameters:

| Parameter | Description | Value |
|---|---|---|
| CANNo | CAN bus number | 0 ... CANMAX |
| iType | Command selection | See table below. |

| iType | Description: Value |
|---|---|
| CAN_CMD_HEARTBEAT | Only the own heartbeat status is changed. Selecting heartbeat states:<br>CAN_HEARTBEAT_STOPPED (0x04)<br>CAN_HEARTBEAT_OPERATIONAL (0x05)<br>CAN_HEARTBEAT_PREOPERATIONAL (0x7F) |
| CAN_CMD_NMT | The heartbeat status is changed for all other devices or for a specific device on the CAN bus. Selecting heartbeat states (NMT master):<br>CAN_NMT_OPERATIONAL (0x01) or CAN_NMT_START (0x01)<br>CAN_NMT_STOP (0x02)<br>CAN_NMT_PREOPERATIONAL (0x80)<br>CAN_NMT_RESET (0x81)<br>CAN_NMT_RESETCOMMUNICATION (0x82) |
| CAN_CMD_TIME_CONSUMER | This command lets you set the device to ready-to-receive state to allow time synchronization via CAN bus (CAN ID 0x100). Refer to document by CiA e.V. DS301 V402 *Selecting Synchronization*, page 59.<br>CAN_TIME_CONSUMER_DISABLE = 0<br>CAN_TIME_CONSUMER_ENABLE = 1 |
| CAN_CMD_TIME_PRODUCER | The time is published on the CAN bus. For more information on the structure refer to document DS301 by CiA e.V., CAN ID 0x100:<br>CAN_TIME_PRODUCER_SEND = 1 (for sending TIME_OF_DAY once) |

**Note**

The macro function `CAN_CMD_NMT_Value(NodeID, CAN_CMD_NMT)` is used to select the command CAN_CMD_NMT.

Values from 0 to 127 are permitted for the node ID parameter. 1 to 127 is the node ID for a specific device. If the command is to be sent to all devices on the CAN bus, use the parameter `CAN_CMD_NMT_ALLNODES(0)`.

**CANNo parameter**

This parameter specifies the number of the CAN interface. CANNo = 0 is assigned to the first interface. The number of CAN interfaces depends on the device. For information on the maximum number of CAN interfaces (CANMAX) refer to the chapters *Technical Specifications* and *Quick Reference* in the corresponding manual.

**Return value**

This function sends the following return values to the higher-level program.

| Return value | |
|---|---|
| 0 | OK |
| -1 | Error when checking parameters<br>Command not known |

**How to use this function (example 1)**

Task: Set the own heartbeat status to **operational**.

```
Result := CanOpenSetCommand(0, CAN_CMD_HEARTBEAT,
CAN_HEARTBEAT_OPERATIONAL);
```

**How to use this function (example 2)**

Task: Set the own heartbeat status and the status of all other devices on the CAN bus to **operational**.

```
Result := CanOpenSetCommand(0, CAN_CMD_NMT,
CAN_CMD_NMT_Value(CAN_CMD_NMT_ALLNODES, CAN_NMT_OPERATIONAL));
```

**How to use this function (example 3)**

Task: Set the heartbeat status of the device with the node ID 60 (0x3C) to **operational**.

```
Result := CanOpenSetCommand(0, CAN_CMD_NMT, CAN_CMD_NMT_Value(60,
CAN_NMT_OPERATIONAL));
```

**How to use this function (example 4)**

Task: Enable time synchronization via CAN bus (CAN ID 0x100).

```
Result := CanOpenSetCommand(0, CAN_CMD_TIME_CONSUMER,
CAN_TIME_CONSUMER_ENABLE);
```

**How to use this function (example 5)**

Task: Publish the time on the CAN bus.

```
Result := CanOpenSetCommand(0, CAN_CMD_TIME_PRODUCER,
CAN_TIME_PRODUCER_SEND);
```

# STX function: CanOpenUploadSDO()

**Introduction**

The function `CanOpenUploadSDO()` lets you access a particular object in the object directory of the message recipient and read the value of the object.

Data is exchanged in accordance with the SDO upload protocol. Supported transfer types are **segmented** (more than 4 data bytes) and **expedited** (up to 4 data bytes).

**Function declaration**

```
Function CanOpenUploadSDO(
    CANNo:Int,              // Number of the bus line
    NodeID:Int,             // Device ID
    wIndex:Word,
    SubIndex:Byte,
    DataType:Int,           // Type of the data to be received
    // Data length for the global variable DataAddr
    DataLength:Int,
    // Global variable into which the received value is entered
    const ref DataAddr,
    ref Busy: Int,          // Status of the SDO transmission
) :Int;
```

**Function parameters**

The `CanOpenUploadSDO()` function has got the following parameters:

| Parameter | Description | Value |
|---|---|---|
| CANNo | CAN bus number | 0 ... CANMAX |
| NodeID | Node ID of the message recipient | 1 ... 127 |
| wIndex | Index number of the object | 0 ... 0xFFFF |
| SubIndex | Subindex number of the object | 0 ... 255 |
| DataType | Type of data to be received | 2 ... 27 |
| DataLength | Data length of the global variable DataAddr | |
| DataAddr | Global variable into which the received value is to be entered | |
| Busy | Status of the SDO transmission | |

**Return value**

This function sends the following return values to the higher-level program.

| Return value | |
|---|---|
| 0 | OK |
| -1 | Error in checking parameters |
| -2 | Device in **Stop** status |
| -3 | DataType is greater than DataLength |
| -4 | Insufficient memory |

**CANNo parameter**

This parameter specifies the number of the CAN interface. CANNo = 0 is assigned to the first interface. The number of CAN interfaces depends on the device. For information on the maximum number of CAN interfaces (CANMAX) refer to the chapters *Technical Specifications* and *Quick Reference* in the corresponding manual.

**DataType parameter**

The following data types can be received.

| Byte types | CANopen® format | Jetter format |
|---|---|---|
| 1 | CANOPEN_INTEGER8<br>CANOPEN_UNSIGNED8 | Byte |
| 2 | CANOPEN_INTEGER16<br>CANOPEN_UNSIGNED16 | Word |
| 3 | CANOPEN_INTEGER24<br>CANOPEN_UNSIGNED24 | - |
| 4 | CANOPEN_INTEGER32<br>CANOPEN_UNSIGNED32<br>CANOPEN_REAL | Int |
| 5 | CANOPEN_INTEGER40<br>CANOPEN_UNSIGNED40 | - |
| 6 | CANOPEN_INTEGER48<br>CANOPEN_UNSIGNED48<br>CANOPEN_TIME_OF_DAY<br>CANOPEN_TIME_DIFFERENCE | - |
| 7 | CANOPEN_INTEGER56<br>CANOPEN_UNSIGNED46 | - |
| 8 | CANOPEN_INTEGER64<br>CANOPEN_UNSIGNED64<br>CANOPEN_REAL64 | - |
| n | CANOPEN_VISIBLE_STRING<br>CANOPEN_OCTET_STRING<br>CANOPEN_UNICODE_STRING<br>CANOPEN_DOMAIN | String |

**Busy parameter**

After successfully calling up the function, the **Busy** parameter is set to SDOACCESS_INUSE. With an error in transmission, **Busy** is set to SDOACCESS_ERROR. With a successful transmission, the function returns the number of bytes transmitted.

**Busy - Error codes**

With an error in transmission, **Busy** returns an error code. The following error codes are available:

**SDOACCESS_STILLUSED**

Another task is communicating with the same node ID.

**SDOACCESS_TIMEOUT**

The task has been timed out because the device with the specified node ID is not responding.

If the specified node ID does not respond within 1 second, the timeout bit is set.

**SDOACCESS_ILLCMD**

The response to the request is invalid.

**SDOACCESS_ABORT**

Access to the device with the specified node ID was aborted.

**SDOACCESS_SYSERROR**

General internal error

**Macro definitions**

The following macros have been defined in connection with this function:

**SDOACCESS_FINISHED (busy)**

This macro checks whether communication has finished.

**SDOACCESS_ERROR (busy)**

This macro checks whether an error has occurred.

# STX function: CanOpenDownloadSDO()

**Introduction**

The function `CanOpenDownloadSDO()` lets you access a particular object in the Object Directory of the message recipient and specify the value of the object. Data is exchanged in accordance with the SDO upload protocol. Supported transfer types are **segmented** or **block** (more than 4 data bytes) and **expedited** (up to 4 data bytes).

**Function declaration**

```
Function CanOpenDownloadSDO(
    CANNo:Int,              // Number of the bus line
    NodeID:Int,             // Device ID
    wIndex:Word,
    SubIndex:Byte,
    DataType:Int,           // Type of the data to be sent
    // Data length of the global variable DataAddr
    DataLength:Int,
    // Global variable holding the value to be sent
    const ref DataAddr,
    ref Busy: Int,          // Status of the SDO transmission
) :Int;
```

**Function parameters**

The `CanOpenDownloadSDO()` function has got the following parameters:

| Parameter | Description | Value |
|---|---|---|
| CANNo | CAN bus number | 0 ... CANMAX |
| NodeID | Node ID of the message recipient | 1 ... 127 |
| wIndex | Index number of the object | 0 ... 0xFFFF |
| SubIndex | Subindex number of the object | 0 ... 255 |
| DataType | Type of data to be sent | 2 ... 27 |
| DataLength | Data length of the global variable DataAddr | |
| DataAddr | Global variable into which the value to be sent is to be entered | |
| Busy | Status of the SDO transmission | |

**Return value**

This function sends the following return values to the higher-level program.

| Return value | |
|---|---|
| 0 | OK |
| -1 | Error when checking parameters |
| -2 | Device in **Stop** status (own heartbeat status) |
| -3 | DataType is greater than DataLength |
| -4 | Insufficient memory |

**CANNo parameter**

This parameter specifies the number of the CAN interface. CANNo = 0 is assigned to the first interface. The number of CAN interfaces depends on the device. For information on the maximum number of CAN interfaces (CANMAX) refer to the chapters *Technical Specifications* and *Quick Reference* in the corresponding manual.

**DataType parameter**

The following data types can be received.

| Byte types | CANopen® format | Jetter format |
|---|---|---|
| 1 | CANOPEN_INTEGER8<br>CANOPEN_UNSIGNED8 | Byte |
| 2 | CANOPEN_INTEGER16<br>CANOPEN_UNSIGNED16 | Word |
| 3 | CANOPEN_INTEGER24<br>CANOPEN_UNSIGNED24 | - |
| 4 | CANOPEN_INTEGER32<br>CANOPEN_UNSIGNED32<br>CANOPEN_REAL | Int |
| 5 | CANOPEN_INTEGER40<br>CANOPEN_UNSIGNED40 | - |
| 6 | CANOPEN_INTEGER48<br>CANOPEN_UNSIGNED48<br>CANOPEN_TIME_OF_DAY<br>CANOPEN_TIME_DIFFERENCE | - |
| 7 | CANOPEN_INTEGER56<br>CANOPEN_UNSIGNED46 | - |
| 8 | CANOPEN_INTEGER64<br>CANOPEN_UNSIGNED64<br>CANOPEN_REAL64 | - |
| n | CANOPEN_VISIBLE_STRING<br>CANOPEN_OCTET_STRING<br>CANOPEN_UNICODE_STRING<br>CANOPEN_DOMAIN | String |

**Busy parameter**

After successfully calling up the function, the **Busy** parameter is set to SDOACCESS_INUSE. With an error in transmission, **Busy** is set to SDOACCESS_ERROR. With a successful transmission, the function returns the number of bytes transmitted.

**"Busy" error codes**

With an error in transmission, Busy returns an error code. The following error codes are available:

**SDOACCESS_STILLUSED**
Another task is communicating with the same node ID.

**SDOACCESS_TIMEOUT**
The task has been timed out because the device with the given node ID is not responding.
If the specified node ID does not respond within 1 second, the timeout bit is set.

**SDOACCESS_ILLCMD**
The response to the request is invalid.

**SDOACCESS_ABORT**
Access to the device with the specified node ID was aborted.

**SDOACCESS_BLKSIZEINV**
Communication error with Block Download

**SDOACCESS_SYSERROR**
General internal error

**Macro definitions**

The following macros have been defined in connection with this function:

**SDOACCESS_FINISHED (busy)**
This macro checks whether communication has finished.

**SDOACCESS_ERROR (busy)**
This macro checks whether an error has occurred.

# STX function: CanOpenAddPDORx()

**Introduction**

The function `CanOpenAddPDORx()` lets you specify which process data, sent by other CANopen® devices, must be received.

Process data can be received only when a CANopen® device is sending them.

**Notes**

- Only if the CANopen® devices on the bus are in state **operational**, the PDO telegram is transmitted.
- The smallest time unit for the event time is 1 ms.
- The smallest time unit for the inhibit time is 1 ms.

**Function declaration**

```
Function CanOpenAddPDORx(
    CANNo:Int,              // Number of the bus line
    CANID:Int,              // CAN identifier
    // Starting position of data to be received
    BytePos:Int,
    DataType:Int,           // Data type of the data to be received
    // Data length of the global variable VarAddr
    DataLength:Int,
    // Global variable into which the received value is entered
    const ref VarAddr,
    // Cycle time for receiving a telegram
    // Event time
    EventTime: Int,
    // Minimum interval between two received messages
    // Inhibit time
    InhibitTime: Int,
    Paramset: Int,          // Bit-coded parameter
) :Int;
```

**Function parameters**

The `CanOpenAddPDORx()` function has got the following parameters:

| Parameter | Description | Value |
|---|---|---|
| CANNo | CAN bus number | 0 ... CANMAX |
| CANID | CAN identifier 11-bit<br>CAN identifier 29-bit | 0 ... 0x7FF<br>0 ... 0x1FFFFFFF |
| BytePos | Starting position of data to be received | 0 ... 7 |
| DataType | Data type of data to be received | 2 ... 13, 15 ... 27 |
| DataLength | Data length of the global variable VarAddr | |
| VarAddr | Global variable into which the received value is entered | |
| EventTime | Time lag between two telegrams (> InhibitTime) | |

| Parameter | Description | Value |
|---|---|---|
| InhibitTime | Minimum time lag between two telegrams received (< EventTime) | |
| Paramset | Bit-coded parameter | |

**Return value**

This function sends the following return values to the higher-level program.

| Return value | |
|---|---|
| 0 | OK |
| -1 | Error when checking parameters |
| -3 | DataType is greater than DataLength |
| -4 | Insufficient memory |

**CANNo parameter**

This parameter specifies the number of the CAN interface. CANNo = 0 is assigned to the first interface. The number of CAN interfaces depends on the device. For information on the maximum number of CAN interfaces (CANMAX) refer to the chapters *Technical Specifications* and *Quick Reference* in the corresponding manual.

**CANID parameter**

The **CANID** parameter is used to transfer the CAN identifier. The CAN identifier is generated with a macro. The CAN identifier depends on the node ID of the other communicating user and on whether it is a PDO1, PDO2, PDO3 or PDO4 message.

**Macro definitions:**

```
#Define CANOPEN_PDO1_RX (NodeID)    ((NodeID) + 0x180)
#Define CANOPEN_PDO2_RX (NodeID)    ((NodeID) + 0x280)
#Define CANOPEN_PDO3_RX (NodeID)    ((NodeID) + 0x380)
#Define CANOPEN_PDO4_RX (NodeID)    ((NodeID) + 0x480)

#Define CANOPEN_PDO1_TX (NodeID)    ((NodeID) + 0x200)
#Define CANOPEN_PDO2_TX (NodeID)    ((NodeID) + 0x300)
#Define CANOPEN_PDO3_TX (NodeID)    ((NodeID) + 0x400)
#Define CANOPEN_PDO4_TX (NodeID)    ((NodeID) + 0x500)
```

**Example for calling up the macro:**

CANOPEN_PDO2_RX (64)

⇨ The resulting CAN identifier is: 2C0h = 40h + 280h

**Default CAN identifier distribution**

For CANopen® the following CAN identifier distribution is predefined. In this case, the node number is embedded in the identifier.

| 11-bit identifier (binary) | Identifier (decimal) | Identifier (hexadecimal) | Description |
|---|---|---|---|
| 000000000000 | 0 | 0 | Network management |
| 000100000000 | 128 | 80h | Synchronization |
| 0001xxxxxxxx | 129 - 255 | 81h - FFh | Emergency |
| 0011xxxxxxxx | 385 - 511 | 181h - 1FFh | PDO1 (tx) |
| 0100xxxxxxxx | 513 - 639 | 201h - 27Fh | PDO1 (rx) |
| 0101xxxxxxxx | 641 - 767 | 281h - 2FFh | PDO2 (tx) |
| 0110xxxxxxxx | 769 - 895 | 301h - 37Fh | PDO2 (rx) |
| 0111xxxxxxxx | 897 - 1023 | 381h - 3FFh | PDO3 (tx) |
| 1000xxxxxxxx | 1025 - 1151 | 401h -47Fh | PDO3 (rx) |
| 1001xxxxxxxx | 1153 - 1279 | 481h - 4FFh | PDO4 (tx) |
| 1010xxxxxxxx | 1281 - 1407 | 501h - 57Fh | PDO4 (rx) |
| 1011xxxxxxxx | 1409 - 1535 | 581h - 5FFh | Send SDO |
| 1100xxxxxxxx | 1537 - 1663 | 601h - 67Fh | Receive SDO |
| 1110xxxxxxxx | 1793 - 1919 | 701h - 77Fh | NMT error control |
| xxxxxxxx = Node number 1 - 127 | | | |

**DataType parameter**

The following data types can be received.

| Byte types | CANopen® format | Jetter format |
|---|---|---|
| 1 | CANOPEN_INTEGER8<br>CANOPEN_UNSIGNED8 | Byte |
| 2 | CANOPEN_INTEGER16<br>CANOPEN_UNSIGNED16 | Word |
| 3 | CANOPEN_INTEGER24<br>CANOPEN_UNSIGNED24 | - |
| 4 | CANOPEN_INTEGER32<br>CANOPEN_UNSIGNED32<br>CANOPEN_REAL | Int |
| 5 | CANOPEN_INTEGER40<br>CANOPEN_UNSIGNED40 | - |
| 6 | CANOPEN_INTEGER48<br>CANOPEN_UNSIGNED48<br>CANOPEN_TIME_OF_DAY<br>CANOPEN_TIME_DIFFERENCE | - |
| 7 | CANOPEN_INTEGER56<br>CANOPEN_UNSIGNED46 | - |
| 8 | CANOPEN_INTEGER64<br>CANOPEN_UNSIGNED64<br>CANOPEN_REAL64 | - |
| n | CANOPEN_VISIBLE_STRING<br>CANOPEN_OCTET_STRING<br>CANOPEN_UNICODE_STRING<br>CANOPEN_DOMAIN | String |

**Paramset parameter**

The following parameters can be transferred to the function. Several parameters can be linked together using the Or function.

**CANOPEN_ASYNCPDORTRONLY**

Receive asynchronous PDOs by sending an RTR frame to the sender (after each expired EventTime). If there is no response to RTR frames, the request time increases to five times the EventTime.

**CANOPEN_ASYNCPDO**

Receive asynchronous PDOs.

**CANOPEN_PDOINVALID**

PDO not received. Disk space is reserved.

**CANOPEN_NORTR**

PDO cannot be requested by RTR (Remote Request).

Only if CANOPEN_ASYNCPDORTROnly has been set, an RTR is sent.

**CANOPEN_29BIT**

Use 29-bit identifier
Default: 11-bit identifier

# STX function: CanOpenAddPDOTx()

**Introduction**

By calling up the `CanOpenAddPDOTx()` function, process data can be deposited on the bus.

However, that should not mean that other CANopen® devices on the bus can also read this process data.

**Notes**

- Only if the CANopen® devices on the bus are in state **operational**, the PDO telegram is transmitted.
- As soon as there are any changes to the process data, another PDO telegram is transmitted immediately.
- The smallest time unit for the event time is 1 ms.
- The smallest time unit for the inhibit time is 1 ms.
- Any unused bytes of a telegram are sent as null.

**Function declaration**

```
Function CanOpenAddPDOTx(
    CANNo:Int,              // Number of the bus line
    CANID:Int,              // CAN identifier
    BytePos:Int,            // Starting position of the data to be sent
    DataType:Int,           // Data type of the data to be sent
    // Data length of the global variable VarAddr
    DataLength:Int,
    // Global variable holding the value to be sent
    const ref VarAddr,
    // Cycle time for sending a telegram
    // Event time
    EventTime: Int,
    // Minimum interval between two transmitted messages
    // Inhibit time
    InhibitTime: Int,
    Paramset: Int,          // Bit-coded parameter
) :Int;
```

**Function parameters**

The `CanOpenAddPDOTx()` function has got the following parameters:

| Parameter | Description | Value |
|---|---|---|
| CANNo | CAN bus number | 0 ... CANMAX |
| CANID | CAN identifier 11-bit<br>CAN identifier 29-bit | 0 ... 0x7FF<br>0 ... 0x1FFFFFFF |
| BytePos | Starting position of data to be sent | 0 ... 7 |
| DataType | Data type of data to be sent | 2 ... 13, 15 ... 27 |
| DataLength | Data length of the global variable VarAddr | |
| VarAddr | Global variable into which the value to be sent is entered | |

| Parameter | Description | Value |
|---|---|---|
| EventTime | Time lag between two telegrams (> InhibitTime) | |
| InhibitTime | Minimum time lag between two telegrams to be sent (< EventTime) | |
| Paramset | Bit-coded parameter | |

**Return value**

This function sends the following return values to the higher-level program.

| Return value | |
|---|---|
| 0 | OK |
| -1 | Error when checking parameters |
| -3 | DataType is greater than DataLength |
| -4 | Insufficient memory |

**CANNo parameter**

This parameter specifies the number of the CAN interface. CANNo = 0 is assigned to the first interface. The number of CAN interfaces depends on the device. For information on the maximum number of CAN interfaces (CANMAX) refer to the chapters *Technical Specifications* and *Quick Reference* in the corresponding manual.

**CANID parameter**

The **CANID** parameter is used to transfer the CAN identifier. The CAN identifier is generated with a macro. The CAN identifier depends on the node ID of the other communicating user and on whether it is a PDO1, PDO2, PDO3 or PDO4 message.

**Macro definitions:**

```
#Define CANOPEN_PDO1_RX (NodeID)    ((NodeID) + 0x180)
#Define CANOPEN_PDO2_RX (NodeID)    ((NodeID) + 0x280)
#Define CANOPEN_PDO3_RX (NodeID)    ((NodeID) + 0x380)
#Define CANOPEN_PDO4_RX (NodeID)    ((NodeID) + 0x480)

#Define CANOPEN_PDO1_TX (NodeID)    ((NodeID) + 0x200)
#Define CANOPEN_PDO2_TX (NodeID)    ((NodeID) + 0x300)
#Define CANOPEN_PDO3_TX (NodeID)    ((NodeID) + 0x400)
#Define CANOPEN_PDO4_TX (NodeID)    ((NodeID) + 0x500)
```

**Example for calling up the macro:**

CANOPEN_PDO2_RX (64)

⇨ The resulting CAN identifier is: 2C0h = 40h + 280h

**Default CAN identifier distribution**

For CANopen® the following CAN identifier distribution is predefined. In this case, the node number is embedded in the identifier.

| 11-bit identifier (binary) | Identifier (decimal) | Identifier (hexadecimal) | Description |
|---|---|---|---|
| 000000000000 | 0 | 0 | Network management |
| 000100000000 | 128 | 80h | Synchronization |
| 0001xxxxxxxx | 129 - 255 | 81h - FFh | Emergency |
| 0011xxxxxxxx | 385 - 511 | 181h - 1FFh | PDO1 (tx) |
| 0100xxxxxxxx | 513 - 639 | 201h - 27Fh | PDO1 (rx) |
| 0101xxxxxxxx | 641 - 767 | 281h - 2FFh | PDO2 (tx) |
| 0110xxxxxxxx | 769 - 895 | 301h - 37Fh | PDO2 (rx) |
| 0111xxxxxxxx | 897 - 1023 | 381h - 3FFh | PDO3 (tx) |
| 1000xxxxxxxx | 1025 - 1151 | 401h -47Fh | PDO3 (rx) |
| 1001xxxxxxxx | 1153 - 1279 | 481h - 4FFh | PDO4 (tx) |
| 1010xxxxxxxx | 1281 - 1407 | 501h - 57Fh | PDO4 (rx) |
| 1011xxxxxxxx | 1409 - 1535 | 581h - 5FFh | Send SDO |
| 1100xxxxxxxx | 1537 - 1663 | 601h - 67Fh | Receive SDO |
| 1110xxxxxxxx | 1793 - 1919 | 701h - 77Fh | NMT error control |
| xxxxxxxx = Node number 1 - 127 | | | |

**DataType parameter**

The following data types can be received.

| Byte types | CANopen® format | Jetter format |
|---|---|---|
| 1 | CANOPEN_INTEGER8<br>CANOPEN_UNSIGNED8 | Byte |
| 2 | CANOPEN_INTEGER16<br>CANOPEN_UNSIGNED16 | Word |
| 3 | CANOPEN_INTEGER24<br>CANOPEN_UNSIGNED24 | - |
| 4 | CANOPEN_INTEGER32<br>CANOPEN_UNSIGNED32<br>CANOPEN_REAL | Int |
| 5 | CANOPEN_INTEGER40<br>CANOPEN_UNSIGNED40 | - |
| 6 | CANOPEN_INTEGER48<br>CANOPEN_UNSIGNED48<br>CANOPEN_TIME_OF_DAY<br>CANOPEN_TIME_DIFFERENCE | - |
| 7 | CANOPEN_INTEGER56<br>CANOPEN_UNSIGNED46 | - |
| 8 | CANOPEN_INTEGER64<br>CANOPEN_UNSIGNED64<br>CANOPEN_REAL64 | - |

| Byte types | CANopen® format | Jetter format |
|---|---|---|
| n | CANOPEN_VISIBLE_STRING<br>CANOPEN_OCTET_STRING<br>CANOPEN_UNICODE_STRING<br>CANOPEN_DOMAIN | String |

**Paramset parameter**

The following parameters can be transferred to the function. Several parameters can be linked together using the Or function.

**CANOPEN_ASYNCPDORTRONLY**
Send asynchronous PDOs by receiving an RTR frame.
This feature is not yet supported at the moment.

**CANOPEN_ASYNCPDO**
Send asynchronous PDO.

**CANOPEN_PDOINVALID**
PDO not sent. The required disk space is reserved.

**CANOPEN_NORTR**
PDO cannot be requested by RTR (Remote Request).

**CANOPEN_29BIT**
Use 29-bit identifier
Default: 11-bit identifier

# Heartbeat monitoring

**Introduction**

The heartbeat protocol is for monitoring the activity of communication partners. If the inactivity exceeds the set interval (Heartbeat consumer time), the status is set to **offline**.

The application program lets you define heartbeat functions, such as

- Displaying information to the user
- Rebooting the device
- Ignoring process data

**Prerequisites**

**Heartbeat monitoring** is available only for specific devices and its availability depends on the OS version, for further details refer to the quick reference on the respective device.

**Registers for heartbeat monitoring**

Heartbeat monitoring uses the following registers:

| Register | Description | Data type | Attributes |
|---|---|---|---|
| 40x001 | Own heartbeat status of the device;<br>Value range:<br>0 = Bootup<br>4 = Stopped<br>5 = Operational<br>127 = Preoperational<br>255 = Offline (default value) | Int | ro (read only) |
| 40x100 | The heartbeat status of all monitored node IDs has changed. Value range:<br>0 = False<br>1 = True | Bool | rw (read and write) |
| 40x101 ... 40x227 | Heartbeat status of nodes with ID 1 ... 127; value range:<br>0 = Bootup<br>4 = Stopped<br>5 = Operational<br>127 = Preoperational<br>255 = Offline (default value) | Byte | ro |
| 40x229 ... 40x355 | Heartbeat timeout of nodes with ID 1 ... 127; value range:<br>0 ... 65535 [ms] | Word | rw |

In the register number, the letter **x** represents the number of the CAN bus line used: x = 0 ... CANMAX.

**Launching heartbeat monitoring**

To launch heartbeat monitoring, proceed as follows:

| Step | Action |
|---|---|
| **1** | Enable heartbeat monitoring:<br>Enter the timeout value into the corresponding register. This value must range between 1 and 65535 [ms]. Example:<br>For CAN 0 and node ID 1: Enter a timeout value of 3000 [ms] into register 400229. |
| **2** | Define in your application program how the device is to respond to individual values in the heartbeat status register.<br>When the state in register 40x101 ... 40x227 changes, the value in register 40x100 changes to 1 (true). |
| **3** | Reset the value in register 40x100 to zero (false).<br>This step ensures that subsequent changes in register 40x101 ... 40x227 can be displayed. |

Heartbeat monitoring starts on receipt of the first heartbeat (including bootup message). The DLC (Data Length Code) of the heartbeat message must be 1.

**Terminating heartbeat monitoring**

To terminate heartbeat monitoring, proceed as follows:

| Step | Action |
|---|---|
| **1** | Disable heartbeat monitoring:<br>Enter a timeout value of 0 [ms] into the timeout register. |

**Emergency message**

When a heartbeat timeout is detected, an emergency message is sent automatically.

On receipt of the next heartbeat message, the emergency message is reset.

**Example:**

The following emergency message is tripped:

| Reference | Value |
|---|---|
| Error code | 0x8130 |
| Error Register | 0x81 |
| Manufacturer error | 0x00,NodeID,0x00,0x00,0x00 |

The message on the CAN bus looks as shown below:

- Own NodeID 5
- Monitored NodeID 1
- ID: 0x85 DLC = 8 Data: 0x30 0x81 0x81 0x00 0x01 0x00 0x00 0x00

**Emergency message Rx**

The declaration of the emergency message Rx consists of the following elements:

```
CanOpenAddEmergencyRx(
    CANNo:Int,         // Number of the bus line
    NodeID:Int,        // Node ID
    // Status, number of valid messages
    ref stCanOpenEmergencyStat:CanOpenEmergencyStat,
    // Array holding the emergency messages
    ref CanOpenEmergencyMSG:CanOpenEmergencyArray,
    ):int
```

**Example:**

The above program lines must be included into the corresponding tasks of your application program. The example below shows an emergency message from a device with node ID 21.

```
...
// Initializing the CAN bus once.

...

// Defining global variables
Var
    stCanOpenEmergencyMsg : ARRAY[5] of CanOpenEmergencyMsg;
    stCanOpenEmergencyStat : CanOpenEmergencyStat;
End_Var;

stCanOpenEmergencyStat.lBuffer := sizeof(stCanOpenEmergencyMsg);
iRet:= CanOpenAddEmergencyRx(0,                          // CANNo.
                             21,                         // NodeID
                             stCanOpenEmergencyStat,     // Status
                             stCanOpenEmergencyMsg);     // Array

...
```

**The above program lines produce the following result:**

When the device with node ID 21 receives an emergency message, the value in register 400100 switches from 0 to 1 (true).

Reset this value always to 0 (false). In doing so, you make sure that new emergency messages are displayed.

**Emergency message Tx**

The declaration of the emergency message Tx consists of the following elements:

```
CanOpenAddEmergencyTx(
    // Number of the bus line
    CANNo:int,
    // For error code see CiA DS 301 V4.02 page 60
    // or CiA DS 4xx (device profile)
    ErrorCode:word,
    // Error register (object 0x1001)
    ErrorRegister:byte,
    // 5 bytes can be used at the user's discretion
    ManufacturerArray:ByteArray5,
    // True = An error has occurred
    // False = Error has been cleared (acknowledged)
    bSet:bool
    ):Int;
```

## CANopen® object dictionary

**Supported objects**

The operating system of the CANopen® devices supports the following objects:

| Index (hex) | Object (code) | Object name | Type | Attributes |
|---|---|---|---|---|
| 1000 | VAR | Device type | Unsigned32 | ro (read only) |
| 1001 | VAR | Error register | Unsigned8 | ro |
| 1002 | VAR | Manufacturer status | Unsigned32 | ro |
| 1003 | ARRAY | Pre-defined error field | Unsigned32 | ro |
| 1008 | VAR | Manufacturer device name | String | const |
| 1009 | VAR | Manufacturer hardware version | String | const |
| 100A | VAR | Manufacturer software version | String | const |
| 100B | VAR | Node ID | Unsigned32 | ro |
| 1017 | VAR | Producer heartbeat time | Unsigned16 | rw (read & write) |
| 1018 | RECORD | Identity | Identity | ro |
| 1200 | RECORD | Server 1 - SDO parameter | SDO parameter | ro |
| 1201 | RECORD | Server 2 - SDO parameter | SDO parameter | rw |
| 1203 | RECORD | Server 3 - SDO parameter | SDO parameter | rw |
| 1203 | RECORD | Server 4 - SDO parameter | SDO parameter | rw |

**Device Type object (index 0x1000)**

The structure of the **Device Type object** is shown in the following table.

| Index | Subindex | Default | Description |
|---|---|---|---|
| 0x1000 | 0 | 0x0000012D | Device type (read-only) |

### Error Register object (index 0x1001)

The function `CanOpenAddEmergencyTx()` lets you set the bits in this register.

The structure of the **Error Register object** is shown in the following table.

| Index | Subindex | Default | Description |
|---|---|---|---|
| 0x1001 | 0 | 0 | Error register (read-only) |

This object implements the CANopen® error register functionality.

The following error messages may appear:

- Bit 0 = Generic error
- Bit 1 = Current error
- Bit 2 = Voltage error
- Bit 3 = Temperature error
- Bit 4 = Communication error (overrun, error state)
- Bit 5 = Specific device profile error
- Bit 6 = Reserved (always 0)
- Bit 7 = Manufacturer-specific error

### Pre-defined Error Field object (index 0x1003)

The structure of the **Pre-defined Error Field object** is shown in the following table.

| Index | Subindex | Default | Description |
|---|---|---|---|
| 0x1003 | 0 | 0 | Number of errors entered in the array's standard error field |
| | 1 | 0 | Most recent error<br>0 indicates no error |
| | 2 ... 254 | - | Earlier errors |

This object shows a history list of errors that have been detected by the device. The maximum length of the list is 254 errors. The list content is deleted on restart.

**Composition of standard error field**

2-byte LSB: Error code

2-byte MSB: Additional information

### Manufacturer Device Name object (index 0x1008)

The structure of the **Manufacturer Device Name object** is shown in the following table.

| Index | Subindex | Default | Description |
|---|---|---|---|
| 0x1008 | 0 | device name | Hardware name |

**Manufacturer Hardware Version object (index 0x1009)**

The structure of the **Manufacturer Hardware Version object** is shown in the following table.

| Index | Subindex | Default | Description |
|---|---|---|---|
| 0x1009 | 0 | | OS version of the device |

**Manufacturer Software Version object (index 0x100A)**

The structure of the **Manufacturer Software Version object** is shown in the following table.

| Index | Subindex | Default | Description |
|---|---|---|---|
| 0x100A | 0 | | Software version of the application program that runs on the device |

The entry in this index is made via the parameter **SWVersion** of the STX function `CanOpenInit()`.

**Node ID object (index 0x100B)**

The structure of the **Node ID object** is shown in the following table.

| Index | Subindex | Default | Description |
|---|---|---|---|
| 0x100B | 0 | | Node ID of the given device |

**Producer Heartbeat Time object (index 0x1017)**

The structure of the **Producer Heartbeat Time object** is shown in the following table.

| Index | Subindex | Default | Description |
|---|---|---|---|
| 0x1017 | 0 | 1,000 [ms] | Heartbeat time |

**CANopen® registers**

The table below lists the device registers associated with the CANopen® Object Dictionary.

The letter x in the register number represents the CAN bus number ranging from 0 ... CANMAX.

| Register number | Description | Value range | Attributes | Data type |
|---|---|---|---|---|
| 40x000 | Own node ID | 1 ... 127 | rw (read & write) | Int |
| 40x001 | Own heartbeat status | 0 = Bootup<br>4 = Stopped<br>5 = Operational<br>127 = Preoperational<br>255 = Offline | ro (read only) | Int |
| 40x002 | | Refer to object 0x1001 | ro | Int |
| 40x019 | | | ro | Int (IP format) |

| Register number | Description | Value range | Attributes | Data type |
|---|---|---|---|---|
| 40x020 | | | rw | Int |
| 40x021 | | | rw | Int |
| 40x022 | | | rw | Int |
| 40x023 | | | rw | Int |
| 40x030 | | | rw | Int |
| 40x100 | | | rw | bool |
| 40x400 | | | rw | bool |
| 40x101 ... 40x227 | Node ID 1 ... 127 Status | 0 = Bootup<br>4 = Stopped<br>5 = Operational<br>127 = Preoperational<br>255 = Offline (default) | ro | byte |
| 40x229 ... 40x355 | Node ID 1 ... 127 timeout | 0 ... 65535 ms | rw | word |

# 10.19 User-programmable CAN-Prim interface

**CAN-Prim interface**

The user-programmable CAN-Prim interface allows to send and receive CAN messages. The CAN messages are completely processed in the application program.

**CAN-Prim - The benefit**

This feature is not only apt for CANopen® devices. The customer can rather communicate with third-party devices which are based on a CAN protocol.

**Applications**

The user-programmable CAN-Prim interface can be used for the following applications:

- Devices which are equipped with a CAN interface can be controlled via proprietary protocols.
- Controlling of CANopen® capable devices
- ...

**Required programmer's skills**

To be able to program user-programmable CAN-Prim interfaces basic knowledge of Controller Area Networks (CAN) is required. These are some of them:

- Structure of CAN messages
- CANopen® features

**Contents**


| Topic | Page |
|---|---|
| Restrictions regarding the CAN-Prim interface | 553 |
| User-programmable CAN-Prim interface - Operating principle | 557 |
| Internal processes of the CAN-Prim interface | 558 |
| Register description - CAN-Prim interface | 559 |
| CAN message box - Description of registers for direct access | 564 |
| CAN message box - Description of registers for indirect access | 570 |
| Using the CAN-Prim interface | 574 |
| Using CAN-ID masks | 577 |
| RTR frames via CAN-Prim interface | 578 |

## Restrictions regarding the CAN-Prim interface

**Restrictions regarding connectable modules**

When using the user-programmable CAN-Prim interface, the following restrictions apply:

- If 29-bit CAN identifiers are used, the serial number of non-intelligent JX2-I/O module must start with 2.

**CAN messages during boot phase**

Between launching the JC-360 and starting the application program (boot phase of the JX2 system bus), the connected CAN modules are not permitted to send any CAN messages.

**Time response**

The interval between two CAN messages received via CAN interface should be at least 10 ms. In case of shorter time intervals, the JC-360 is not able to process all CAN messages for CAN-Prim reception.

If several CAN messages of the same CAN-ID are to be received, an application program featuring a high reaction and processing speed is required to prevent buffer overflows (overrun-bit). Adjusting the task switch procedure and task prioritization (TASKPRIORITY) do not necessarily grant processing all CAN messages.

**Earmarked CAN-IDs**

When peripheral modules are simultaneously operated on the JX2 system bus and the CAN-Prim interface, certain CAN-IDs are earmarked.

| Modules on the JX2 system bus | Earmarked CAN-IDs |
|---|---|
| For all modules | 0x100, 0x701, 0x702, 0x703, 0x704, 0x705, 0x706, 0x707, 0x708, 0x709, 0x70A, 0x732, 0x733, 0x734, 0x735, 0x736, 0x737, 0x738, 0x739, 0x73A, 0x73B, 0x746, 0x747, 0x748, 0x749, 0x74A, 0x74B, 0x74C, 0x74D, 0x74E, 0x74F |
| JX2-I/O modules | 0x180, 0x181, 0x182, 0x183, 0x184, 0x185, 0x186, 0x187, 0x188, 0x189, 0x18A, 0x18B, 0x18C, 0x18D, 0x18E, 0x18F, 0x190, 0x191, 0x192, 0x193, 0x194, 0x195, 0x196, 0x197, 0x198, 0x199, 0x19A, 0x19B, 0x19C, 0x19D, 0x19E, 0x19F, 0x1A0, 0x1A1, 0x1A2, 0x1A3, 0x1A4, 0x1A5, 0x1A6, 0x1A7, 0x1A8, 0x1A9, 0x1AA, 0x1AB, 0x1AC, 0x1AD, 0x1AE, 0x1AF, 0x1B0, 0x1B1, 0x1B2, 0x1B3, 0x1B4, 0x1B5, 0x1B6, 0x1B7, 0x1B8, 0x1B9, 0x1BA, 0x1BB, 0x1BC, 0x1BD, 0x1BE, 0x1BF, 0x380, 0x381, 0x382, 0x383, 0x384, 0x385, 0x386, 0x387, 0x388, 0x389, 0x38A, 0x38B, 0x38C, 0x38D, 0x38E, 0x38F, 0x390, 0x391, 0x392, 0x393, 0x394, 0x395, 0x396, 0x397, 0x398, 0x399, 0x39A, 0x39B, 0x39C, 0x39D, 0x39E, 0x39F, 0x3A0, 0x3A1, 0x3A2, 0x3A3, 0x3A4, 0x3A5, 0x3A6, 0x3A7, 0x3A8, 0x3A9, 0x3AA, 0x3AB, 0x3AC, 0x3AD, 0x3AE, 0x3AF, 0x3B0, 0x3B1, 0x3B2, 0x3B3, 0x3B4, 0x3B5, 0x3B6, 0x3B7, 0x3B8, 0x3B9, 0x3BA, 0x3BB, 0x3BE, 0x3BF |

| Modules on the JX2 system bus | Earmarked CAN-IDs |
|---|---|
| JX2 slave modules | 0x081, 0x082, 0x083, 0x084, 0x085, 0x086, 0x087, 0x088, 0x089, 0x08A, 0x08B, 0x08C, 0x08D, 0x08E, 0x08F, 0x090, 0x09F, 0x0A0, 0x0A1, 0x0A2, 0x0A3, 0x0A4, 0x0A5, 0x0A6, 0x0A7, 0x0A8, 0x0A9, 0x0AA, 0x0AB, 0x0AC, 0x0AD, 0x0AE, 0x0AF, 0x161, 0x162, 0x163, 0x164, 0x165, 0x166, 0x167, 0x168, 0x169, 0x16A, 0x16B, 0x16C, 0x16D, 0x16E, 0x16F, 0x1D1, 0x1D2, 0x1D3, 0x1D4, 0x1D5, 0x1D6, 0x1D7, 0x1D8, 0x1D9, 0x1DA, 0x1DB, 0x1DC,0x1DD, 0x1DE, 0x1DF |
| JX3 modules | 0x180, 0x181, 0x182, 0x183, 0x184, 0x185, 0x186, 0x187, 0x188, 0x189, 0x18A, 0x18B, 0x18C, 0x18D, 0x18E, 0x18F, 0x190, 0x191, 0x192, 0x193, 0x194, 0x195, 0x196, 0x197, 0x198, 0x199, 0x19A, 0x19B, 0x19C, 0x19D, 0x19E, 0x19F, 0x1A0, 0x1A1, 0x1A2, 0x1A3, 0x1A4, 0x1A5, 0x1A6, 0x1A7, 0x1A8, 0x1A9, 0x1AA, 0x1AB, 0x1AC, 0x1AD, 0x1AE, 0x1AF, 0x1B0, 0x1B1, 0x1B2, 0x1B3, 0x1B4, 0x1B5, 0x1B6, 0x1B7, 0x1B8, 0x1B9, 0x1BA, 0x1BB, 0x1BC, 0x1BD, 0x1BE, 0x1BF, 0x320, 0x321, 0x322, 0x323, 0x324, 0x325, 0x326, 0x327, 0x328, 0x329, 0x32A, 0x32B, 0x32C, 0x32D, 0x32E, 0x32F, 0x330, 0x331, 0x332, 0x333, 0x334, 0x335, 0x336, 0x337, 0x338, 0x339, 0x33A, 0x33B, 0x33C, 0x33D, 0x33E, 0x380, 0x381, 0x382, 0x383, 0x384, 0x385, 0x386, 0x387, 0x388, 0x389, 0x38A, 0x38B, 0x38C, 0x38D, 0x38E, 0x38F, 0x390, 0x391, 0x392, 0x393, 0x394, 0x395, 0x396, 0x397, 0x398, 0x399, 0x39A, 0x39B, 0x39C, 0x39D, 0x39E, 0x39F, 0x3A0, 0x3A1, 0x3A2, 0x3A3, 0x3A4, 0x3A5, 0x3A6, 0x3A7, 0x3A8, 0x3A9, 0x3AA, 0x3AB, 0x3AC, 0x3AD, 0x3AE, 0x3AF, 0x3B0, 0x3B1, 0x3B2, 0x3B3, 0x3B4, 0x3B5, 0x3B6, 0x3B7, 0x3B8, 0x3B9, 0x3BA, 0x3BB, 0x3BE, 0x3BF, 0x3E0, 0x3E1, 0x3E2, 0x3E3, 0x3E4, 0x3E5, 0x3E6, 0x3E7, 0x3E8, 0x3E9, 0x3EA, 0x3EB, 0x3EC, 0x3ED, 0x3EE, 0x3EF, 0x3F0, 0x3F1, 0x3F2, 0x3F3, 0x3F4, 0x3F5, 0x3F6, 0x3F7, 0x3F8, 0x3F9, 0x3FA, 0x3FB, 0x3FC, 0x3FD, 0x3FE |

| Modules on the JX2 system bus | Earmarked CAN-IDs |
|---|---|
| JX-SIO and<br>CANopen® modules | 0x1C6, 0x1C7, 0x1C8, 0x1C9, 0x1CA, 0x1CB, 0x1CC, 0x1CD, 0x1CE, 0x1CF, 0x246, 0x247, 0x248, 0x249, 0x24A, 0x24B, 0x24C, 0x24D, 0x24E, 0x24F, 0x2C6, 0x2C7, 0x2C8, 0x2C9, 0x2CA, 0x2CB, 0x2CC, 0x2CD, 0x2CE, 0x2CF, 0x346, 0x347, 0x348, 0x349, 0x34A, 0x34B, 0x34C, 0x34D, 0x34E, 0x34F, 0x3C6, 0x3C7, 0x3C8, 0x3C9, 0x3CA, 0x3CB, 0x3CC, 0x3CD, 0x3CE, 0x3CF, 0x446, 0x447, 0x448, 0x449, 0x44A, 0x44B, 0x44C, 0x44D, 0x44E, 0x44F, 0x4C6, 0x4C7, 0x4C8, 0x4C9, 0x4CA, 0x4CB, 0x4CC, 0x4CD, 0x3CE, 0x4CF, 0x581, 0x582, 0x583, 0x584, 0x585, 0x586, 0x587, 0x588, 0x589, 0x58A, 0x5B2, 0x5B3, 0x5B4, 0x5B5, 0x5B6, 0x5B7, 0x5B8, 0x5B9, 0x5BA, 0x5BB, 0x5C6, 0x5C7, 0x5C8, 0x5C9, 0x5CA, 0x5CB, 0x5CC, 0x5CD, 0x5CE, 0x5CF, 0x601, 0x602, 0x603, 0x604, 0x605, 0x606, 0x607, 0x608, 0x609, 0x60A, 0x632, 0x633, 0x634, 0x635, 0x636, 0x637, 0x638, 0x639, 0x63A, 0x63B, 0x646, 0x647, 0x648, 0x649, 0x64A, 0x64B, 0x64C, 0x64D, 0x64E, 0x64F, 0x732, 0x733, 0x734, 0x735, 0x736, 0x737, 0x738, 0x739, 0x73A, 0x73B, 0x746, 0x747, 0x748, 0x749, 0x74A, 0x74B, 0x74C, 0x74D, 0x74E, 0x74F |
| Festo CP-FB modules | 0x010, 0x110, 0x120, 0x130, 0x140, 0x150, 0x1E0, 0x1F0, 0x250, 0x260, 0x270, 0x350, 0x360, 0x370, 0x3B0 |
| LioN-S modules | 0x2E0, 0x2E1, 0x2E2, 0x2E3, 0x2E4, 0x2E5, 0x2E6, 0x2E7, 0x2E8, 0x2E9, 0x2EA, 0x2EB, 0x2EC, 0x2ED, 0x2EE, 0x2EF, 0x2F0, 0x2F1, 0x2F2, 0x2F3, 0x2F4, 0x2F5, 0x2F6, 0x2F7, 0x2F8, 0x2F9, 0x2FA, 0x2FB, 0x2FC, 0x2FD, 0x2FE, 0x360, 0x361, 0x362, 0x363, 0x364, 0x365, 0x366, 0x367, 0x368, 0x369, 0x36A, 0x36B, 0x36C, 0x36D, 0x36E, 0x36F, 0x370, 0x371, 0x372, 0x373, 0x374, 0x375, 0x376, 0x377, 0x378, 0x379, 0x37A, 0x37B, 0x37C, 0x37D, 0x37E, 0x581, 0x582, 0x583, 0x584, 0x585, 0x586, 0x587, 0x588, 0x589, 0x58A, 0x58B, 0x58C, 0x58D, 0x58E, 0x58F, 0x590, 0x591, 0x592, 0x593, 0x594, 0x595, 0x596, 0x597, 0x598, 0x599, 0x59A, 0x59B, 0x59C, 0x59D, 0x59E, 0x59F, 0x5A0, 0x601, 0x602, 0x603, 0x604, 0x605, 0x606, 0x607, 0x608, 0x609, 0x60A, 0x60B, 0x60C, 0x60D, 0x60E, 0x60F, 0x610, 0x611, 0x612, 0x613, 0x614, 0x615, 0x616, 0x617, 0x618, 0x619, 0x61A, 0x61B, 0x61C, 0x61D, 0x61E, 0x61F, 0x620, 0x701, 0x702, 0x703, 0x704, 0x705, 0x706, 0x707, 0x708, 0x709, 0x70A, 0x70B, 0x70C, 0x70D, 0x70E, 0x70F, 0x710, 0x711, 0x712, 0x713, 0x714, 0x715, 0x716, 0x717, 0x718, 0x719, 0x71A, 0x71B, 0x71C, 0x71D, 0x71E, 0x71F, 0x720 |

| Modules on the JX2 system bus | Earmarked CAN-IDs |
|---|---|
| BWU1821 | 0x281, 0x282, 0x283, 0x284, 0x285, 0x286, 0x287, 0x288, 0x289, 0x28A, 0x28B, 0x28C, 0x28D, 0x28E, 0x28F, 0x290, 0x291, 0x292, 0x293, 0x294, 0x295, 0x296, 0x297, 0x298, 0x299, 0x29A, 0x29B, 0x29C, 0x29D, 0x29E, 0x29F, 0x301, 0x302, 0x303, 0x304, 0x305, 0x306, 0x307, 0x308, 0x309, 0x30A, 0x30B, 0x30C, 0x30D, 0x30E, 0x30F, 0x310, 0x311, 0x312, 0x313, 0x314, 0x315, 0x316, 0x317, 0x318, 0x319, 0x31A, 0x31B, 0x31C, 0x31D, 0x31E, 0x31F, 0x481, 0x482, 0x483, 0x484, 0x485, 0x486, 0x487, 0x488, 0x489, 0x48A, 0x48B, 0x48C, 0x48D, 0x48E, 0x48F, 0x490, 0x491, 0x492, 0x493, 0x494, 0x495, 0x496, 0x497, 0x498, 0x499, 0x49A, 0x49B, 0x49C, 0x49D, 0x49E, 0x49F, 0x501, 0x502, 0x503, 0x504, 0x505, 0x506, 0x507, 0x508, 0x509, 0x50A, 0x50B, 0x50C, 0x50D, 0x50E, 0x50F, 0x510, 0x511, 0x512, 0x513, 0x514, 0x515, 0x516, 0x517, 0x518, 0x519, 0x51A, 0x51B, 0x51C, 0x51D, 0x51E, 0x51F, 0x5C6, 0x5C7, 0x5C8, 0x5C9, 0x5CA, 0x5CB, 0x5CC, 0x5CD, 0x5CE, 0x5CF, 0x646, 0x647, 0x648, 0x649, 0x64A, 0x64B, 0x64C, 0x64D, 0x64E, 0x64F, 0x746, 0x747, 0x748, 0x749, 0x74A, 0x74B, 0x74C, 0x74D, 0x74E, 0x74F |
| LJX7-CSL | 0x481, 0x482, 0x483, 0x484, 0x485, 0x486, 0x487, 0x488, 0x489, 0x48A, 0x48B, 0x48C, 0x48D, 0x48E, 0x48F, 0x490, 0x491, 0x492, 0x493, 0x494, 0x495, 0x496, 0x497, 0x498, 0x499, 0x49A, 0x49B, 0x49C, 0x49D, 0x49E, 0x49F, 0x501, 0x502, 0x503, 0x504, 0x505, 0x506, 0x507, 0x508, 0x509, 0x50A, 0x50B, 0x50C, 0x50D, 0x50E, 0x50F, 0x510, 0x511, 0x512, 0x513, 0x514, 0x515, 0x516, 0x517, 0x518, 0x519, 0x51A, 0x51B, 0x51C, 0x51D, 0x51E, 0x51F, 0x581, 0x582, 0x583, 0x584, 0x585, 0x586, 0x587, 0x588, 0x589, 0x58A, 0x58B, 0x58C, 0x58D, 0x58E, 0x58F, 0x590, 0x591, 0x592, 0x593, 0x594, 0x595, 0x596, 0x597, 0x598, 0x599, 0x59A, 0x59B, 0x59C, 0x59D, 0x59E, 0x59F, 0x5A0, 0x601, 0x602, 0x603, 0x604, 0x605, 0x606, 0x607, 0x608, 0x609, 0x60A, 0x60B, 0x60C, 0x60D, 0x60E, 0x60F, 0x610, 0x611, 0x612, 0x613, 0x614, 0x615, 0x616, 0x617, 0x618, 0x619, 0x61A, 0x61B, 0x61C, 0x61D, 0x61E, 0x61F, 0x620, 0x701, 0x702, 0x703, 0x704, 0x705, 0x706, 0x707, 0x708, 0x709, 0x70A, 0x70B, 0x70C, 0x70D, 0x70E, 0x70F, 0x710, 0x711, 0x712, 0x713, 0x714, 0x715, 0x716, 0x717, 0x718, 0x719, 0x71A, 0x71B, 0x71C, 0x71D, 0x71E, 0x71F, 0x720 |

# User-programmable CAN-Prim interface - Operating principle

**Operating principle**

The user-programmable CAN-Prim interface uses message boxes for data exchange between CAN bus and application program. Each message box is able to accommodate a complete CAN message.

16 message boxes are available to the user. Each of these boxes can be configured either as inbox or as outbox with a specific CAN-ID.

**Technical specifications**

| Function | Description |
|---|---|
| CAN-ID | 11-bit or 29-bit |
| Number of message boxes | 16 |

**Enabling the user-programmable CAN-Prim interface**

The CAN-Prim interface is enabled via Bits im R 200002077 *JX2-system bus - special functions Register description R 200002077* (see page 559).

# Internal processes of the CAN-Prim interface

**Introduction**

The CAN-Prim interface processes the following tasks independently:

- Sending of CAN messages
- Reception of CAN messages
- Filtering of CAN messages on reception

**Internal reception of CAN messages**

The CAN-Prim interface receives new CAN messages in the following way:

| Step | Description | |
|---|---|---|
| **1** | The CAN bus receives a valid CAN message. | |
| **2** | The CAN-ID matches the receiving mask. | |
| **3** | The CAN-ID matches the CAN-ID of a message box which has been configured as inbox. | |
| **4** | **If in R 200010530 + message box number *20 of the message box ...** | **... then ...** |
| | ... bit 1 *NEW-DAT* = 0, | ... bit 1 *NEW-DAT* becomes = 1;<br>proceed with step 5 |
| | ... bit 1 *NEW-DAT* = 1, | ... bit 2 *OVERRUN* becomes = 1;<br>CAN message data are discarded. |
| **5** | R 200010503 *FIFO filling level* is incremented.<br>This register shows whether new CAN messages have been received, as well as the number of messages. | |
| **6** | The message box number is entered into R 200010504 *FIFO data*.<br>This register shows which of the messages boxes has received a new CAN message. | |
| **7** | In R 200010500 *CAN-Prim Status*, bit 1 *NEW DAT* = 1. | |

# Register description - CAN-Prim interface

**Registers for configuring the JX2 system bus**

The CAN-Prim interface is enabled in R 200002077 *JX2-system bus special functions*.

| Registers | Description |
|---|---|
| **R 200002029** | JX2 system bus - Baud rate |
| **R 200002077** | JX2 system bus special functions |

**Registers for configuring the CAN-Prim interface**

| Registers | Description |
|---|---|
| **R 200010500** | CAN-Prim status register |
| **R 200010501** | CAN-Prim command register |
| **R 200010503** | FIFO occupancy - Number of received messages |
| **R 200010504** | FIFO data -<br>Numbers of message boxes which have received new messages |
| **R 200010506** | Global receiving mask |
| **R 200010507** | Global receiving ID |

**R 200002077**

**Enabling JX2 system bus special functions**

The value of this register influences the behavior at initializing of the JX2 system bus.

| **Meaning of the individual bits** | | |
|---|---|---|
| **Bit 3, bit 2** | **Activate CAN-Prim in addition to JX2 system bus** | |
| | 01 = | The CAN-Prim interface and the JX2 system bus are enabled following the next launch of the JX2 system bus. This requires a restart of the controller.<br>This function allows to connect JX2 expansion modules. |
| **Bit 3, bit 2** | **Enable CAN-Prim and CANopen® STX API only.** | |
| | 1x = | At the next restart, the JX2 system bus is not initialized. The CAN-Prim interface can be used.<br>All node IDs can be used **without any** restrictions.<br>The controller does not initialize any JX2 expansion modules on the JX2 system bus. For this reason, JX2 expansion modules **cannot** be connected.<br>The CANopen® STX API can be used. |
| **Bit 4** | **CAN IDCAN IDs 0x081 ... 9x09F for CAN-Prim** | |
| | 1 = | The CAN-Prim interface allows communication with the CAN IDs 0x081 ... 0x09F.<br>Generally, master-slave operations with JX2 slave modules and MC axes are executed via these CAN IDs, . |

| Module register properties | |
|---|---|
| Value after reset | Non-volatile; factory setting: 0 |
| Takes effect | Next time when the controller is launched |

**R 200010500**

## CAN-Prim status register

R 200010500 allows to evaluate the status of the CAN-Prim interface.

| Meaning of the individual bits | | |
|---|---|---|
| **Bit 1** | **NEW-DAT** | |
| | 1 = | At least one message box has received a new CAN message. |
| **Bit 2** | **Length of CAN ID** | |
| | 0 = | The length of sent/received CAN IDs is 11 bits |
| | 1 = | The length of sent/received CAN IDs is 29 bits |

| Module register properties | |
|---|---|
| Type of access | Read |
| Takes effect | When the CAN-Prim interface is enabled |

**R 200010501**

## CAN-Prim command register

R 200010501 is used to transfer certain commands to the CAN-Prim interface.

| CAN-Prim interface - Commands | |
|---|---|
| **7** | **Clear the FIFO buffer**<br>This command is for clearing all entries in the FIFO buffer.<br>Result: R 200010503 = 0 |
| **8** | **Set the standard ID length to 11 bits**<br>The ID length for all CAN messages is set to 11 bits.<br>Result:<br>Bit 2 = 0 in R 200010500<br>R 200010506 = 0<br>R 200010507 = 0<br>R 200010542 + message box number *20 := 0x7FF (in all message boxes) |

| CAN-Prim interface - Commands | |
|---|---|
| **9** | **Set the standard ID length to 29 bits**<br>The ID length for all CAN messages is set to 29 bits.<br>Result:<br>Bit 2 = 1 in R 200010500<br>R 200010506 = 0<br>R 200010507 = 0<br>R 200010542 + message box number *20 := 0x7FFFFFFF (in all message boxes) |
| **10** | **Nachrichtenboxen auf den Empfang neuer Nachrichten prüfen**<br>The CAN-Prim interface automatically checks the inbox for new CAN messages. Command 10 forces manual checking of pending messages.<br>By now, issuing command 10 is not obligatory any more. |

| Module register properties | |
|---|---|
| Takes effect | When the CAN-Prim interface is enabled |

**R 200010503**

## FIFO buffer occupancy

R 200010503 shows if further CAN messages have been received, as well as the number of messages.

Subtracting the number read first from the number read next renders the number of new messages received.

| Module register properties | | |
|---|---|---|
| Values | Number of received messages: | 0 ... 16 |
| Type of access | Read | |
| Takes effect | When the CAN-Prim interface is enabled | |

**R 200010504**

## FIFO data

R 200010504 shows which of the messages boxes has received the latest new CAN message. Read access to R 200010504 removes the value which has been read last from the FIFO buffer. This access decrements the value of R 200010503 by one.

**Note:**

Each read access to this register, even via an active JetSym setup screen, decrements the number of received CAN messages.

| **Module register properties** | | |
|---|---|---|
| Values | No FIFO data available: | -1 |
| | Number of the message box containing new data: | 0 ... 15 |
| Type of access | Read access removes characters | |
| Value after reset | -1 | |
| Takes effect | When the CAN-Prim interface is enabled | |

**R 200010506**

## Global receiving mask

The global receiving mask is for filtering the bits of the received CAN IDs. If the bit of the global receiving mask is set, the received bit of the CAN ID is compared with the global receiving ID as shown in R 200010507.

| **Module register properties** | | |
|---|---|---|
| Values | In the case of 11-bit CAN IDs | 0 ... 0x7FF |
| | In the case of 29-bit CAN IDs | 0 ... 0x1FFFFFFF |
| Bit = 0 | Bit is not compared with R 200010507 | |
| Bit = 1 | Bit is compared with R 200010507 | |
| Takes effect | When the CAN-Prim interface is enabled | |

**R 200010507**

## Global receiving ID

The global receiving ID and R 200010506 *Global receiving mask* are for setting a CAN ID range which is then forwarded to the CAN-Prim interface.

| **Module register properties** | | |
|---|---|---|
| Values | In the case of 11-bit CAN IDs | 0 ... 0x7FF |
| | In the case of 29-bit CAN IDs | 0 ... 0x1FFFFFFF |
| Takes effect | When the CAN-Prim interface is enabled | |

# CAN message box - Description of registers for direct access

**Direct access**

For programming purposes, always use registers for direct access to message boxes. 20 registers with identical functions are assigned to each message box. The registers of individual message boxes start from a certain basic register number.

| Message box number | Basic register number |
|---|---|
| 0 | R 200010530 |
| 1 | R 200010550 |
| 2 | R 200010570 |
| 3 | R 200010590 |
| 4 | R 200010610 |
| 5 | R 200010630 |
| 6 | R 200010650 |
| 7 | R 200010670 |
| 8 | R 200010690 |
| 9 | R 200010710 |
| 10 | R 200010730 |
| 11 | R 200010750 |
| 12 | R 200010770 |
| 13 | R 200010790 |
| 14 | R 200010810 |
| 15 | R 200010830 |

**Registers for message boxes of the CAN-Prim interface**

20 registers with identical functions are assigned to each message box. The register number of individual message boxes is calculated from the basic register number and the message box number (0 ... 15).

| Register | Description |
|---|---|
| **R 200010530 + message box number *20** | Message box status register |
| **R 200010531 + message box number *20** | Message box configuration register |
| **R 200010532 + message box number *20** | CAN-ID |
| **R 200010533 + message box number *20** | Number of data bytes |
| **R 200010534 + message box number *20** | Data byte 0 |
| **R 200010535 + message box number *20** | Data byte 1 |

| Register | Description |
|---|---|
| **R 200010536 + message box number *20** | Data byte 2 |
| **R 200010537 + message box number *20** | Data byte 3 |
| **R 200010538 + message box number *20** | Data byte 4 |
| **R 200010539 + message box number *20** | Data byte 5 |
| **R 200010540 + message box number *20** | Data byte 6 |
| **R 200010541 + message box number *20** | Data byte 7 |
| **R 200010542 + message box number *20** | CAN-ID mask |
| **R 200010543 + message box number *20** | Box command register |
| **R 200010544 + message box number *20** | Received CAN-ID |
| **R 200010545 + message box number *20** | Not used |
| **R 200010546 + message box number *20** | Not used |
| **R 200010547 + message box number *20** | Not used |
| **R 200010548 + message box number *20** | Not used |
| **R 200010549 + message box number *20** | Not used |

**R 200010530 +message box number *20**

## Message box status register

This register shows the status of the message box.

| Meaning of the individual bits | | |
|---|---|---|
| **Bit 0** | **Valid** | |
| | 1 = | The message box is enabled |
| **Bit 1** | **NEW-DAT** | |
| | 1 = | The message box has received a CAN message. Reception of additional CAN messages is blocked. |
| **Bit 2** | **OVERRUN** | |
| | 1 = | A new CAN message for this message box was being received while bit 1 *NEW-DAT* was = 1.<br>The new message is discarded. |

| Meaning of the individual bits | |
|---|---|
| **Bit 3** | **Sending error**<br>1 = An error has occurred when sending a CAN message from this message box. |

| Module register properties | |
|---|---|
| Type of access | Read |
| Takes effect | When the CAN-Prim interface is enabled |

**R 200010543 +message box number *20**

## Box command register

R 200010543 + message box number *20 is used to transfer certain commands to the message box.

| CAN-Prim interface - Commands | |
|---|---|
| **1** | **Enabling the message box**<br>The message box is enabled. When enabling the message box, the system checks whether the CAN-ID of the message box has been reserved by the JX2 system bus or not.<br>**Result, if the CAN-ID has not been reserved:**<br>Bit 0 = 1 in R 200010530 + message box number *20 |
| **2** | **Disabling the message box**<br>The message box is disabled.<br>**Result:**<br>Bit 0 = 0 in R 200010530 + message box number *20 |
| **3** | **Sending CAN messages**<br>A CAN message is sent. |
| **4** | **Clearing the NEW DAT bit**<br>Clears bit 1 *NEW-DAT* in R 200010530 + message box number *20.<br>The message box is able to receive CAN messages again.<br>**Result:**<br>Bit 1 = 0 in R 200010530 + message box number *20<br>If for all message boxes the NEW DAT bit is 0, bit 1 = 0 in R 200010500. |
| **5** | **Clearing the OVERRUN bit**<br>Clears bit 2 *OVERRUN* in R 200010530 + message box number *20 of the message box.<br>**Result:**<br>Bit 2 = 0 in R 200010530 + message box number *20 |
| **6** | **Clearing the sending error bit**<br>Clears bit 3 *Transmit error* in R 200010530 + message box number *20 of the message box.<br>**Result:**<br>Bit 3 = 0 in R 200010530 + message box number *20 |

| Module register properties | |
|---|---|
| Takes effect | When the CAN-Prim interface is enabled |

**R 200010531 +message box number *20**

## Message box configuration register

R 200010531 + message box number *20 is for configuring the message box.

| Configuration values | |
|---|---|
| **0** | **Inbox**<br>For configuring the message box as inbox |
| **1** | **Outbox**<br>For configuring the message box as outbox for standard frames |
| **2** | **Outbox RTR**<br>For configuring the message box as outbox for RTR frames |

| Module register properties | |
|---|---|
| Takes effect | When the CAN-Prim interface is enabled |

**R 200010532 + message box number *20**

## CAN-ID

In the case of an outbox, a CAN message is sent using the CAN-ID.

In the case of an inbox, CAN messages with this CAN-ID - which is masked by the CAN-ID mask - are received.

| Module register properties | | |
|---|---|---|
| Values | In the case of 11-bit CAN-IDs | 0 ... 0x7FF |
| | In the case of 29-bit CAN-IDs | 0 ... 0x1FFFFFFF |
| Takes effect | When the CAN-Prim interface is enabled and the message box is disabled, i.e. if bit 0 = 0 in R 200010530 + message box *20 | |

**R 200010542 + message box number *20**

## CAN-ID mask

The CAN-ID mask can be used to configure which bits of a received CAN-ID are compared with the configured CAN-ID of the message box.

| Module register properties | | |
|---|---|---|
| Values | Bit = 0 | Bit is not compared with CAN-ID |
| | Bit = 1 | Bit is compared with CAN-ID |
| Takes effect | When the CAN-Prim interface is enabled | |

**R 200010544 + message box number *20**

## Received CAN-ID

In the case of an inbox, the CAN-IDs of received CAN messages are entered here.

| Module register properties | | |
|---|---|---|
| Type of access | Read | |
| Values | In the case of 11-bit CAN-IDs | 0 ... 0x7FF |
| | In the case of 29-bit CAN-IDs | 0 ... 0x1FFFFFFF |
| Takes effect | When the CAN-Prim interface is enabled | |

**R 200010533 + message box number *20**

## Number of data bytes

In the case of an outbox, a CAN message is sent with this number of data bytes.

In the case of an inbox, the number of received data bytes is entered.

| Module register properties | | |
|---|---|---|
| Values | Number of data bytes: | 0 ... 8 |
| Takes effect | When the CAN-Prim interface is enabled | |

**R 200010534 ...**
**R 200010541 + message box number *20**

**Data bytes 0 through 7**

In the case of an outbox, a CAN message is sent with these data bytes.
In the case of an inbox, the received data bytes are entered.

| **Module register properties** | | |
|---|---|---|
| Values | Data of data bytes: | 0 ... 255 |
| Takes effect | When the CAN-Prim interface is enabled | |

# CAN message box - Description of registers for indirect access

**Indirect access**

To get indirect access to message boxes of the CAN-Prim interface always select the message box using R 200010502 *Message Box Number*.

To allow compatibility with previous OS versions the registers for indirect access are still supported. **Always use the registers for direct access when programming the CAN-Prim interface.**

**R 200010501**

## CAN-Prim command register

R 200010501 is used to transfer certain commands to the CAN-Prim interface.

| CAN-Prim interface - Commands | |
|---|---|
| **1** | **Enable the message box**<br>The selected message box in R 200010502 is enabled. When enabling the message box, the system checks whether the CAN-ID of the message box has been reserved by the system bus or not.<br>**Result:**<br>Bit 0 = 1 in R 200010510 |
| **2** | **Disable the message box**<br>The selected message box in R 200010502 is disabled.<br>**Result:**<br>Bit 0 = 0 in R 200010510 |
| **3** | **Send a CAN message**<br>A CAN message is sent containing the data of the selected message box. |
| **4** | **Clear the NEW DAT bit**<br>This command is for clearing bit 1 *NEW-DAT* in R 200010510 which enables the selected message box to receive CAN messages again.<br>**Result:**<br>Bit 1 = 0 in R 200010510 |
| **5** | **Clear the OVERRUN bit**<br>Clears bit 2 *OVERRUN* in R 200010510 of the selected message box.<br>**Result:**<br>Bit 2 = 0 in R 200010510 |
| **6** | **Clear the sending error bit**<br>Clears bit 3 **Transmit error** in R 200010510 of the selected message box.<br>**Result:**<br>Bit 3 = 0 in R 200010510 |
| **7** | **Clear the FIFO buffer**<br>This command is for clearing all entries in the FIFO buffer.<br>**Result:**<br>R 200010503 = 0 |

| CAN-Prim interface - Commands | |
|---|---|
| **8** | **Set the standard ID length to 11 bits**<br>The ID length for all CAN messages is set to 11 bits.<br>**Result:**<br>Bit 2 = 0 in R 200010500<br>R 200010506 := 0<br>R 200010507 := 0 |
| **9** | **Set the standard ID length to 29 bits**<br>The ID length for all CAN messages is set to 29 bits.<br>**Result:**<br>Bit 2 = 1 in R 200010500<br>R 200010506 := 0<br>R 200010507 := 0 |
| **10** | **Check message boxes for receiving new messages**<br>The CAN-Prim interface automatically checks the inbox for new CAN messages. Command 10 forces manual checking of pending messages.<br>By now, issuing command 10 is not obligatory any more. |

| Module register properties | |
|---|---|
| Takes effect | When the CAN-Prim interface is enabled |

**R 200010502**

## Message box number

R 200010502 is for selecting a message box. The data contained in the message box can then be accessed via R 200010510 through R 200010521.

| Module register properties | | |
|---|---|---|
| Values | Message box number: | 0 ... 15 |
| Takes effect | When the CAN-Prim interface is enabled | |

**R 200010510**

## Message box status register

This register shows the status of the message box.

| Meaning of the individual bits | | |
|---|---|---|
| **Bit 0** | **Valid** | |
| | 1 = | The message box is enabled |
| **Bit 1** | **NEW-DAT** | |
| | 1 = | The message box has received a CAN message. Reception of additional CAN messages is blocked. |
| **Bit 2** | **OVERRUN** | |
| | 1 = | A new CAN message for this message box was being received while bit 1 *NEW-DAT* was = 1.<br>The new message is discarded. |
| **Bit 3** | **Sending error** | |
| | 1 = | An error has occurred when sending a CAN message from this message box. |

| Module register properties | |
|---|---|
| Type of access | Read |
| Takes effect | When the CAN-Prim interface is enabled |

**R 200010511**

## Message box configuration register

R 200010511 is for configuring the message box.

| Configuration values | |
|---|---|
| **0** | **Inbox**<br>For configuring the message box as inbox |
| **1** | **Outbox**<br>For configuring the message box as outbox for standard frames |
| **2** | **Outbox RTR**<br>For configuring the message box as outbox for RTR frames |

| Module register properties | |
|---|---|
| Takes effect | When the CAN-Prim interface is enabled |

**R 200010512**

## CAN-ID

In the case of an outbox, a CAN message is sent using the CAN-ID.
In the case of an inbox, only CAN messages with this CAN-ID are received.

| Module register properties | | |
|---|---|---|
| Values | In the case of 11-bit CAN-IDs | 0 ... 0x7FF |
| | In the case of 29-bit CAN-IDs | 0 ... 0x1FFFFFFF |
| Takes effect | When the CAN-Prim interface is enabled and the message box is disabled, i.e. if bit 0 = 0 in R 200010510 | |

**R 200010513**

## Number of data bytes

In the case of an outbox, a CAN message is sent with this number of data bytes.
In the case of an inbox, the number of received data bytes is entered.

| Module register properties | | |
|---|---|---|
| Values | Number of data bytes: | 0 ... 8 |
| Takes effect | When the CAN-Prim interface is enabled | |

**R 200010514 ...**
**R 200010521**

## Data bytes 0 through 7

In the case of an outbox, a CAN message is sent with these data bytes.
In the case of an inbox, the received data bytes are entered.

| Module register properties | | |
|---|---|---|
| Values | Data of data bytes: | 0 ... 255 |
| Takes effect | When the CAN-Prim interface is enabled | |

# Using the CAN-Prim interface

**Initialization**

To initialize the CAN-Prim interface proceed as follows:

| Step | Action | |
|---|---|---|
| **1** | Set bit 2 = 1 in R 20002077 *JX2 system bus special functions*.<br>If JX2 expansion modules have also been connected to the JX2 system bus, then bit 2 = 1. | |
| **2** | Start up the JX2 system bus. | |
| **3** | Configure the CAN ID length for all message boxes. | |
| | **If the CAN ID length...** | **... then ...** |
| | ... is 11 bits, | ... command register 200010501 := 8; |
| | ... is 29 bits, | ... command register 200010501 := 9; |

**Configuring a message box as outbox**

To configure a message box as outbox, proceed as follows:

| Step | Action |
|---|---|
| **1** | Select a message box. In this manual message box 1 is used (basic register number 200010550). |
| **2** | Configure message box 1 as outbox:<br>R 200010551 := 1; |
| **3** | Configure the CAN ID for sending messages<br>R 200010552 := CAN ID; |
| **4** | Activate message box 1:<br>R 200010563 := 1;<br>**Result if configuration was successful:**<br>Bit 0 = 1 in R 200010550 |

**Sending a CAN message**

To send a CAN message proceed as follows:

| Step | Action |
|---|---|
| **1** | Select a message box. In this manual message box 1 is used. |
| **2** | Enter the number of data bytes to be sent:<br>R 200010553 := Number of bytes; |
| **3** | Enter the content into the data bytes to be sent:<br>R 200010554 := Data byte 0;<br>R 200010555 := Data byte 1;<br>...<br>R 200010561 := Data byte 7; |
| **4** | Start transmission of the CAN message:<br>R 200010563 := 3;<br>**Result if sending was successful:**<br>Bit 3 = 0 in R 200010550 |

**Configuring a message box as inbox**

To configure a message box for receiving messages proceed as follows:

| Step | Action |
|---|---|
| **1** | Select a message box. In this manual message box 0 is used (basic register number 200010530). |
| **2** | Configure message box 0 as inbox:<br>R 200010531 := 0; |
| **3** | Configure the CAN ID for receiving messages<br>R 200010532 := CAN ID; |
| **4** | Activate message box 1:<br>R 200010543 := 1;<br>**Result if configuration was successful:**<br>Bit 0 = 1 in R 200010530 |

**Receiving a CAN message**

To receive a CAN message in message box 0, proceed as follows:

| Step | Action | |
|---|---|---|
| **1** | Check bit 1 *NEW-DAT* in R 200010500 | |
| | **If ...** | **... then ...** |
| | Bit 1 *NEW-DAT* = 1 in R 200010500, | ... a CAN message has been received.<br>Proceed with step 2. |
| **2** | Read the number of the message box which has received a new CAN message.<br>Message box number :=R 200010504: | |
| **3** | Check the message box for overflow. | |
| | **If ...** | **... then ...** |
| | ... Bit 2 *OVERRUN* = 1 in R 200010530, | ... an overflow has occurred. |
| **4** | Read the number of received bytes<br>Number of bytes = R 200010533; | |
| **5** | Read the received bytes<br>Data byte 0 = R 200010534;<br>Data byte 1 = R 200010535;<br>...<br>Data byte 7 = R 200010541; | |
| **6** | Acknowledge reception<br>R 200010543 := 4;<br>**Result if message was successfully received:**<br>Bit 1 = 0 in R 200010530 | |

# Using CAN-ID masks

**Introduction**

Usually the CAN-Prim interface receives only CAN messages with a CAN-ID which matches the configured CAN-ID of the message box.

You can use a mask to expand CAN-IDs of a message box which are to be received. Each message box has got a CAN-ID and a CAN-ID mask of its own.

**Functioning principle**

| If ... | ... then ... |
|---|---|
| ... bit = 0 in R 200010542 + message box number *20 | ... the bit of the CAN-ID received is not evaluated. |
| ... bit = 1 in R 200010542 + message box number *20 | ... the bit of the CAN-ID received must match the configured CAN-ID. |

# RTR frames via CAN-Prim interface

**RTR frames**

RTR (Remote Transmission Request) frames are a type of message specific to CAN. Using an RTR frame a CAN node A can prompt another CAN node B to send a message. An RTR frame cannot be used to send user data. Node B is prompted to send a frame of the same CAN-ID and the corresponding data.

**Configuration for sending and receiving RTR frames**

| Step | Action |
|---|---|
| **1** | Select any message box for sending RTR frames and another message box for receiving them.<br>In this manual message box 0 is used for sending and message box 1 for receiving RTR frames. |
| **2** | Configure message box 0 as outbox for RTR frames:<br>R 200010531 := 2; |
| **3** | Configure the CAN-ID of the RTR frame:<br>R 200010532 := CAN-ID; |
| **4** | Activate message box 0:<br>R 200010543 := 1;<br>**Result:**<br>Bit 0 = 1 in R 200010530 |
| **5** | Configure message box 1 as inbox for replies to an RTR frame:<br>R 200010551 := 0; |
| **6** | Configure the CAN-ID of the RTR frame:<br>R 200010552 := CAN-ID; |
| **7** | Activate message box 1:<br>R 200010563 := 1;<br>**Result:**<br>Bit 0 = 1 in R 200010550 |

### Sending and receiving RTR frames

<table>
<tr><th>Step</th><th colspan="2">Action</th></tr>
<tr><td>1</td><td colspan="2">Prompt sending an RTR frame from message box 0:<br>R 200010543 := 3;</td></tr>
<tr><td rowspan="3">2</td><td colspan="2">Wait for a reply to the RTR frame in message box 1:</td></tr>
<tr><td>If ...</td><td>... then ...</td></tr>
<tr><td>... bit 1 NEW-DAT = 1 in R200010550,</td><td>... the controller has received the reply to the RTR frame.<br>Proceed with step 3.</td></tr>
<tr><td>3</td><td colspan="2">Read the number of received bytes<br>Number of bytes = R 200010553;</td></tr>
<tr><td>4</td><td colspan="2">Read the received bytes<br>Data byte 0 = R 200010554;<br>Data byte 1 = R 200010555;<br>...<br>Data byte 7 = R 200010561;</td></tr>
<tr><td>5</td><td colspan="2">Acknowledge reception<br>R 200010563 := 4;</td></tr>
<tr><td>⇨</td><td colspan="2">The message box is again ready to receive.</td></tr>
</table>

# 11 Automatic copying of controller data

**Introduction**

This chapter describes the AutoCopy function which allows to copy data within the controller and/or between the controller and an FTP server, the connected expansion modules and a controller within the network. To this end, you can create a command file which is then stored along with the data to an SD card, for example. This command file is automatically processed by the controller during the boot process.

**Functions within the local file system**

The AutoCopy function executes the following functions:

- Storing registers and flags to a file
- Restoring registers and flags from a file
- Creating directories
- Deleting directories
- Copying files
- Deleting files

**Functions within the file system of an FTP server**

The AutoCopy function executes the following functions:

- Copying files from the FTP server
- Copying files to the FTP server
- Deleting files
- Changing directories
- Creating a directory
- Deleting directories

**Areas of application**

There are the following application scopes for the AutoCopy function:

- Where remote control is not possible
- Where there is no PC on site
- If the operator is not able or should not be allowed to make modifications to the plant

The following actions can be taken using the AutoCopy function:

- Modification to the application program
- Modification to the application data
- Modification to the controller configuration
- Operating system update (controller, modules on the system bus, network devices)
- Duplication of a control system

**Prerequisites**

For automatic copying of controller data, the following prerequisites must be fulfilled:

- The programmer is familiar with the file system.
- The programmer must have basic knowledge in the area of FTP application.
- The JC-340 features the **SD card** option.

**config.ini - Example**

This is an example of a configuration file **config.ini** with an entry *AutoCopyIni*.

```
;<Productname> System Configuration
;Copyright (c) 2009 by Jetter AG, Ludwigsburg, Germany

[IP]
Address     = 192.168. 10.209
SubnetMask  = 255.255.255.  0
DefGateway  =   0.  0.  0.  0
DNSServer   = 192.168. 10.244

[HOSTNAME]
SuffixType  = 0
Name        = JetControl360

[PORTS]
JetIPBase   = 50000
JVMDebug    = 52000

[FILES]
AutoCopyIni = /SD/project_name/autocopy.ini
```

**AutoCopyIni - Note**

- The AutoCopy function only makes sense, if the data to be copied have been stored to the SD card. This means that the root directory is */SD/*.

Since operating system version 1.09 of JC-360 the following applies:

- The file **autocopy.ini** can be stored to any subdirectory.
- Instead of **autocopy.ini**, you can name the file arbitrarily.

**Designation**

In this description, *Complete Name* means the name of the file or directory including the complete path.

**Contents**


| Topic | Page |
|---|---|
| Operating principle | 582 |
| autocopy.ini - Structure | 587 |
| Log file | 600 |
| Data files | 602 |

# 11.1 Operating principle

**Introduction**

This chapter describes how to start and execute the AutoCopy function.

**Contents**


| Topic | Page |
|---|---|
| Activating the AutoCopy feature | 583 |
| Executing AutoCopy commands | 584 |
| Terminating AutoCopy mode | 586 |

## Activating the AutoCopy feature

**Introduction**

The AutoCopy function can only be executed when the controller is booting (i.e. after startup).

**Prerequisites**

You have created the command file and stored it to the respective directory.

If the entry *AutoCopyIni* is not available in the configuration file **config.ini** the name of the command file and of the directory is set as follows:

| | Value | Remarks |
|---|---|---|
| File name | autocopy.ini | All lower case letters |
| Directory | /SD/ | Root directory on the SD card |

Since operating system version 1.09 of JC-360 the following applies:

- Die Datei **autocopy.ini** kann in einem beliebigen Unterverzeichnis des Stammverzeichnisses */SD/* auf der SD-Karte sein.
- Instead of **autocopy.ini**, you can name the file arbitrarily.

In this case, it is prerequisite that the configuration file **config.ini** contains the entry *config.ini*. This entry defines the directory and file name of the command file.

**Activating the AutoCopy feature**

To start the AutoCopy function, proceed as follows:

| Step | Action |
|---|---|
| **1** | Switch the controller off. |
| **2** | Insert the SD card completely into the SD slot. |
| **3** | Set the mode selector to *LOAD* position. |
| **4** | Switch the controller on |
| **5** | Wait for the red LED **D1** to be lit and for the green LED **R** and the yellow LED **SD** to flash slowly by approximately 1 Hz. |
| ⇨ | **Result:** The controller executes the AutoCopy function. |
| **6** | Wait for the red LED **D1** and for the green LED **R** to flash slowly by approximately 1 Hz. |
| ⇨ | **Result:** The AutoCopy function is set. |

# Executing AutoCopy commands

**Introduction**

During the boot process in AutoCopy mode the controller executes the commands contained in the command file.

**Restrictions**

In AutoCopy mode the following restrictions of controller functions apply:

- The controller does not execute the application program.
- Communication with the controller is not possible.
- After terminating the AutoCopy function, restart of the controller is required.

**Executing AutoCopy commands**

The OS of the controller processes the AutoCopy function in the following steps:

| Step | Description |
|---|---|
| **1** | The controller opens the command file of the SD card that is specified by the entry *AutoCopyIni* in the configuration file **/System/config.ini**. |
| **2** | The controller reads the values from section `[OPTIONS]`. |
| **3** | The controller reads the command and its parameters from the section `[COMMAND_1]`, processes it and writes the results, if any, into the log file. |
| **4 ... n** | The controller processes the other commands in ascending order up to the number given in section `[OPTIONS]`. |
| **n+1** | The controller calculates the statistic values for all command results and writes them into the log file. |

**LEDs of the JC-360 in AutoCopy mode**

During the boot process of the controller, the OS status LEDs indicate the following:

| Step | Description | | | | | |
|---|---|---|---|---|---|---|
| **1** | | | | | | |
| | **R** | **E** | **D1** | **D2** | **SD** | **State** |
| |  |  |  |  |  | Reset |
| | | | | | | |
| **2** | | | | | | |
| | **R** | **E** | **D1** | **D2** | **SD** | **State** |
| |  |  |  |  |  | Boot loader is running and is checking the OS. |
| | | | | | | |

**3**

| R | E | D1 | D2 | SD | State |
|---|---|---|---|---|---|
| 1Hz | OFF | OFF | OFF | OFF | The OS reads the settings of the DIP switch on the backplane module and checks if an Ethernet switch exists. |

**4**

| R | E | D1 | D2 | SD | State |
|---|---|---|---|---|---|
| 1Hz | ON | OFF | OFF | OFF | The OS initializes the realtime clock and file system. |

**5**

| R | E | D1 | D2 | SD | State |
|---|---|---|---|---|---|
| 1Hz | ON | ON | OFF | | The OS initializes the modules on the JX3 and JX2 system bus and the SD card. |

**6**

| R | E | D1 | D2 | SD | State |
|---|---|---|---|---|---|
| 1Hz | OFF | ON | OFF | | The command file of the AutoCopy function is being processed. |

**7a**

| R | E | D1 | D2 | SD | State |
|---|---|---|---|---|---|
| 1Hz | OFF | 1Hz | OFF | OFF | AutoCopy function is completed; no errors occurred. |

**7b**

| R | E | D1 | D2 | SD | State |
|---|---|---|---|---|---|
| 1Hz | ON | 1Hz | OFF | OFF | AutoCopy function is completed; errors occurred. |

## Terminating AutoCopy mode

**Introduction**

Only a restart of the controller terminates the AutoCopy mode.

**Prerequisites**

Processing the AutoCopy command is completed.

**Terminating AutoCopy mode**

To terminate the AutoCopy mode, proceed as follows:

| Step | Action |
|---|---|
| **1** | Switch the controller off. |
| **2** | The SD card can now be removed (not required). |
| **3** | Set the mode selector to *RUN* or *STOP* position. |
| **4** | Switch the controller on. |

**Result:** The controller is rebooting.

# 11.2 autocopy.ini - Structure

**Introduction**

This chapter covers the structure of the file **autocopy.ini** and the available commands.

**File structure**

This command file of the AutoCopy function is a text file the entries of which are grouped into several sections.

- In these sections you can set values then used by the AutoCopy function.
- You can insert blank lines as required.
- Introduce comment marks by "!", "#" oder ";".

**Sections**

The command file has two section types:

- In the *[OPTIONS]* section, you can make default settings. This file is unique.
- In the *[COMMAND_#]* section, you can specify the commands that are to be executed. The number of command section is limited to a value of 128.

**Contents**


| Topic | Page |
|---|---|
| Section [OPTIONS] | 588 |
| Command sections | 589 |
| Example of a command file | 597 |

## Section [OPTIONS]

**Introduction**

In the `[OPTIONS]` section, you can make default settings. This section exists only once, preferably at the beginning of the file.

**Example**

```
[OPTIONS]
CommandCount = 14
LogFile      = /SD/autocopy.log
LogAppend    = 1
```

**Elements of this section**

The section consists of the following elements:

| **CommandCount** | |
|---|---|
| In the given example | 14 |
| Function | Number of command sections that follow |
| Allowed values | > = 0 |
| Illegal values | < 0 |
| In case of illegal value or missing entry | 0 |
| **LogFile** | |
| In the given example | /SD/autocopy.log |
| Function | Complete name of the log file |
| Allowed values | ■ All allowed file names<br>■ Directory exists |
| Illegal values | ■ Incorrect filename<br>■ Non-existent directory |
| In case of illegal value or missing entry | The controller does not create a log file. |
| **LogAppend** | |
| In the given example | 1 |
| Function | Defines whether a new log file is to be created or whether it is to be appended to an existing one. |
| Allowed values | ■ 0 = Delete file which may exist and create a new one<br>■ 1 = Append file to an existing one. If no file exists, the controller creates a new log file. |
| Illegal values | ■ < 0<br>■ > 1 |
| In case of illegal value or missing entry | The controller re-creates the log file. |

## Command sections

**Introduction**

In these sections you can specify commands which are then executed by the AutoCopy function.

**Example**

```
[COMMAND_1]
Command        = DirCreate
Path           = /Homepage
ErrorAsWarning = 1

[COMMAND_2]
Command        = FileCopy
Source         = /SD/Index.htm
Destination    = /Homepage/index.htm

[COMMAND_3]
Command        = FtpConnect
ServerAddr     = 192.168.123.45
UserName       = admin
Password       = admin
```

**Section names**

The names of the sections consist of the `COMMAND_` string followed by a value. The value is between one and the value of the `CommandCount` entry from section `[OPTIONS]`.

**Processing commands**

The AutoCopy function processes the commands in order of their section names:

- Starting with the command under section `[COMMAND_1]`
- Ending with the command under the section with the value of entry `CommandCount` from section `[OPTIONS]`
- Each command section may only contain one command. Thus, you have to create an individual section for each command.

**Troubleshooting**

When an error occurs while a command is being processed, the controller makes a corresponding entry in the log file. For each command the user can set, whether the controller is to enter the error into the log file as `Error` or as `Warning`. Make this setting by the optional parameter `ErrorAsWarning`.

| ErrorAsWarning | Entry into the log file |
|---|---|
| Parameter does not exist | Error |
| ErrorAsWarning = 0 | Error |
| ErrorAsWarning = 1 | Warning |

**File names**

- The function parameter for the local file may contain the path to this file, e.g. `'Data/TestFiles/LocalTestFile.txt'`.
- If the file system supports this, the function parameter for the file located on the FTP server can also contain the path to this file. If this feature is not supported, the corresponding directory must be set beforehand using the command `FtpDirChange()`.
- The file system of a JC-360 supports both options.

**Available commands in the local file system**

The following commands are available for access to the local file system:

| **Command = DirCreate** | |
|---|---|
| Function | Creates a subdirectory |
| Parameter name | Path |
| Parameter value | Complete directory name |
| Allowed values | ■ All valid directory names<br>■ Higher-level directories are available |
| Illegal values | ■ Invalid directory name<br>■ Non-existent higher-level directory<br>■ Name of an already existing directory |
| In the event of an illegal value | The controller does not generate the directory. It enters the error into the log file. |
| Example | `[COMMAND_1]`<br>`Command = DirCreate`<br>`Path = /sub1`<br><br>`[COMMAND_2]`<br>`Command = DirCreate`<br>`Path = /sub1/sub2` |

| **Command = DirRemove** | |
|---|---|
| Function | Removes a subdirectory |
| Parameter name | Path |
| Parameter value | Complete directory name |
| Allowed values | ■ All valid directory names<br>■ The directory is empty |
| Illegal values | ■ Invalid directory name<br>■ Directory is not empty |
| In the event of an illegal value | The controller does not delete the directory. It enters the error into the log file. |
| Example | `[COMMAND_8]`<br>`Command = DirRemove`<br>`Path = /sub1/sub2` |

| **Command = FileCopy** | |
|---|---|
| Function | This command is for copying a file. |
| Parameter name 1 | Source |
| Parameter value 1 | Complete name of the source file |

| | |
|---|---|
| Parameter name 2 | Destination |
| Parameter value 2 | Complete name of the destination file |
| Allowed values | ■ All allowed file names<br>■ The destination directory does exist |
| Illegal values | ■ Incorrect filename<br>■ Non-existent source file<br>■ Non-existent destination directory |
| In the event of an illegal value | The controller does not copy the file. It enters the error into the log file. |
| Example | `[COMMAND_1]`<br>`Command = FileCopy`<br>`Source = /SD/OS/JC-360_1.04.0.03.os`<br>`Destination = /System/OS/op_system.os`<br><br>`[COMMAND_2]`<br>`Command = FileCopy`<br>`Source = /SD/Manual.pdf`<br>`Destination = /sub1/Manual.pdf` |

**Command = FileRemove**

| | |
|---|---|
| Function | Deleting a file |
| Parameter name | Path |
| Parameter value | Complete name of the file |
| Allowed values | All allowed file names |
| Illegal values | Incorrect filename |
| In the event of an illegal value | The controller does not delete the file. It enters the error into the log file. |
| Example | `[COMMAND_5]`<br>`Command = FileRemove`<br>`Path = /sub1/Manual.pdf` |

**Command = DaFileRead**

| | |
|---|---|
| Function | Transferring register values and flag states from a data file to the JC-360 |
| Parameter name | DaFile |
| Parameter value | Complete name of the data file |
| Allowed values | All allowed file names for data files |
| Illegal values | ■ Incorrect filename<br>■ Nonexistent data file |
| In the event of an illegal value | The data are not transmitted to the JC-360. The JC-360 enters the error into the log file. |
| Example | `[COMMAND_12]`<br>`Command = DaFileRead`<br>`DaFile = /SD/Data/MyTestData.da` |

**Command = DaFileWrite**

| | |
|---|---|
| Function | This command is for storing register values and flag states to a data file. |
| Parameter name 1 | DaFile |
| Parameter value 1 | Complete name of the data file |
| Allowed values | ■ All allowed file names for data files<br>■ The destination directory does exist |
| Illegal values | ■ Incorrect filename<br>■ Non-existent destination directory |
| In the event of an illegal value | The controller does not generate the data file. It enters the error into the log file. |
| Parameter name 2 | Append |
| Parameter value 2 | Defines whether a new data file is to be created or it is to be appended to an existing one |
| Allowed values | ■ 0 = Delete the data file which may exist and create a new data file<br>■ 1 = Append the file to an existing one. If no file exists, the controller creates a new data file |
| Illegal values | ■ < 0<br>■ > 1 |
| In the event of an illegal value | A new data file will be created |
| Parameter name 3 | Type |
| Parameter value 3 | Defines whether registers or flags are to be stored |
| Allowed values | ■ Registers<br>■ Flag |
| Illegal values | Values other than *Register* or *Flag* |
| In the event of an illegal value | The controller does not generate the data file. It enters the error into the log file. |
| Parameter name 4 | First |
| Parameter value 4 | Number of the first register or flag |
| Allowed values | All valid numbers from the memory area of the corresponding JC-360 |
| Illegal values | Invalid numbers |
| In the event of an illegal value | The controller does not generate the data file. It enters the error into the log file. |
| Parameter name 5 | Last |

| | |
|---|---|
| Parameter value 5 | Number of the last register or flag |
| Allowed values | All valid numbers from the memory area of the corresponding JC-360 which are equal to or greater than the value for *First* |
| Illegal values | ■ Invalid numbers<br>■ Numbers less than *First* |
| In the event of an illegal value | The controller stores only one value (First). |
| Example | |

```
[COMMAND_11]
Command     = DaFileWrite
DaFile      = /SD/MyTestData2.da
Append      = 0
Type        = Register
First       = 1000000
Last        = 1000000

[COMMAND_12]
Command     = DaFileWrite
DaFile      = /SD/MyTestData2.da
Append      = 1
Type        = Flag
First       = 10
Last        = 20

[COMMAND_13]
Command     = DaFileWrite
DaFile      = /SD/MyTestData2.da
Append      = 1
Type        = Register
First       = 1000001
Last        = 1000999
```

## Available commands for access via FTP

The following commands are available for access via network using FTP:

**Command = FtpConnect**

| | |
|---|---|
| Function | Establishing a connection to an FTP server |
| Parameter name 1 | ServerAddr |
| Parameter value 1 | IP address or name of FTP server |
| Allowed values | ■ IP address of the FTP server<br>■ Name which can be resolved through DNS |
| Illegal values | ■ IP address other than that of the FTP server<br>■ Name which cannot be resolved |
| Parameter name 2 | UserName |
| Parameter value 2 | User name for logging on at the FTP server |
| Parameter name 3 | Password |
| Parameter value 3 | Password for logging on at the FTP server |
| In the case of a illegal values | The controller does not establish the connection. It enters the error into the log file. |

| | |
|---|---|
| Example | `[COMMAND_1]`<br>`Command = FtpConnect`<br>`ServerAddr = 192.168.123.45`<br>`UserName = admin`<br>`Password = admin` |
| Restriction | Only one connection with an FTP server can be established at a time.<br>The controller terminates the existing connection, before a connection to another FTP server is established. |

| **Command = FtpFileRead** | |
|---|---|
| Function | Copying file from FTP server into the local file system |
| Parameter name 1 | ServerFile |
| Parameter value 1 | Complete name of the source file in the FTP server |
| Parameter name 2 | ClientFile |
| Parameter value 2 | Complete name of the destination file in the local file system |
| Allowed values | ▪ All allowed file names<br>▪ The destination directory does exist |
| Illegal values | ▪ Incorrect filename<br>▪ Non-existent source file<br>▪ Non-existent destination directory |
| In the event of an illegal value | The controller does not copy the file. It enters the error into the log file. |
| Example | `[COMMAND_8]`<br>`Command = FtpFileRead`<br>`ServerFile = /app/cantest/cantest.es3`<br>`ClientFile = /SD/cantest3.es` |

| **Command = FtpFileWrite** | |
|---|---|
| Function | Copying the file from the local file system into the file system of the FTP server |
| Parameter name 1 | ServerFile |
| Parameter value 1 | Complete name of the destination file in the FTP server |
| Parameter name 2 | ClientFile |
| Parameter value 2 | Complete name of the source file in the local file system |
| Allowed values | ▪ All allowed file names<br>▪ The destination directory does exist |
| Illegal values | ▪ Incorrect filename<br>▪ Non-existent source file<br>▪ Non-existent destination directory |
| In the event of an illegal value | The controller does not copy the file. It enters the error into the log file. |
| Example | `[COMMAND_5]`<br>`Command = FtpFileWrite`<br>`ServerFile = /System/OS/op_system.os`<br>`ClientFile = /SD/OS/JC-360_1.09.0.00.os` |

| **Command = FtpFileRemove** | |
|---|---|
| Funktion | Deleting a file from the FTP server |
| Parameter name | ServerFile |
| Parameter value | Complete filename |
| Allowed values | All allowed file names |
| Illegal values | Incorrect filename |
| In the event of an illegal value | The controller does not delete the file. It enters the error into the log file. |
| Example | `[COMMAND_9]`<br>`Command = FtpFileRemove`<br>`ServerFile = /sub1/Manual.pdf` |

| **Command = FtpDirChange** | |
|---|---|
| Function | Changing the working directory in FTP server |
| Parameter name | ServerDir |
| Parameter value | Complete directory name |
| Allowed values | All valid directory names |
| Illegal values | Invalid directory name |
| In the event of an illegal value | The controller does not switch the directory. It enters the error into the log file. |
| Example | `[COMMAND_12]`<br>`Command = FtpDirChange`<br>`ServerDir = /Data/MyTestData` |

| **Command = FtpDirCreate** | |
|---|---|
| Function | Creating a subdirectory in the FTP server |
| Parameter name | ServerDir |
| Parameter value | Complete directory name |
| Allowed values | ■ All valid directory names<br>■ Higher-level directories are available |
| Illegal values | ■ Invalid directory name<br>■ Non-existent higher-level directory<br>■ Name of an already existing directory |
| In the event of an illegal value | The controller does not generate the directory. It enters the error into the log file. |
| Example | `[COMMAND_6]`<br>`Command = FtpDirCreate`<br>`ServerDir = /Data/MyTestData` |
| Restriction | If a directory with the corresponding path is specified as function parameter, all directories up to the directory to be created must exist. Recursive creation of several directories is not supported. |

| **Command = FtpDirRemove** | |
|---|---|
| Function | Clear the subdirectory in the FTP server |
| Parameter name | ServerDir |
| Parameter value | Complete directory name |

| | |
|---|---|
| Allowed values | ■ All valid directory names<br>■ The directory is empty |
| Illegal values | ■ Invalid directory name<br>■ Directory is not empty |
| In the event of an illegal value | The controller does not delete the directory. It enters the error into the log file. |
| Example | `[COMMAND_8]`<br>`Command = FtpDirRemove`<br>`ServerDir = /Data/MyTestData` |

# Example of a command file

**Task**

The JetControl 340 may serve as an example. Via various JX3 modules, it controls an already existing plant. In this plant, you want to enhance the functions.

To this end, the following modifications are required:

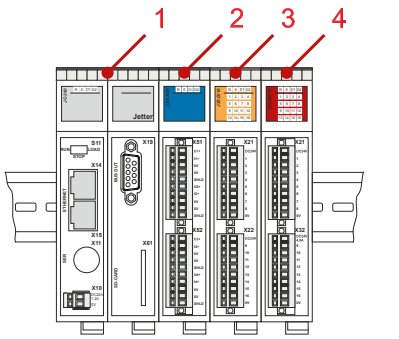
- Operating system update for the controller
- Operating system update for an analog output module
- New application program
- New values for some of the registers

**Solution**

You copy the required files to an SD card and create a command file for the AutoCopy function. Then you send this SD card along with a short instruction sheet to the plant operator. Once the update is completed, the operator is to return the SD card.

**Sample configuration**

This example is based on the following configuration:




| Number | Part | Description |
|---|---|---|
| **1** | JC-340 | Controller |
| **2** | JX3-AO4 | Analog output module<br>I/O module number 2 |
| **3** | JX3-DI16 | Digital input module |
| **4** | JX3-DIO16 | Digital output module |

**SD card contents**

The following illustration shows the directory structure and the files on the SD card from the controller's point of view before the AutoCopy function is executed:




Following execution the log file **autocopy.log** has been added.

**Command file**

```
[OPTIONS]
CommandCount = 7
LogFile      = /SD/autocopy.log
LogAppend    = 0

# update operating system of controller
[COMMAND_1]
Command     = FileCopy
Source      = /SD/OS/JC-340_1.04.0.00.os
Destination = /System/OS/op_system.os

# update operating system of JX3-AO4 module
[COMMAND_2]
Command     = FileCopy
Source      = /SD/OS/JX3-AO4_1.01.0.00.os
Destination = /System/JX3-Module02/OS/system.os

# create user program directories
# probably already present - but to be sure ...
[COMMAND_3]
Command        = DirCreate
Path           = /app
ErrorAsWarning = 1

[COMMAND_4]
Command     = DirCreate
Path        = /app/userprogtest
```

```
# copy user program start file
[COMMAND_5]
Command     = FileCopy
Source      = /SD/UserProgs/start.ini
Destination = /app/start.ini

# copy user program
[COMMAND_6]
Command     = FileCopy
Source      = /SD/UserProgs/userprogtest.es3
Destination = /app/userprogtest/userprogtest.es3

# set registers and flags
[COMMAND_7]
Command     = DaFileRead
DaFile      = /SD/UserData/MyTestData.da
```

# 11.3 Log file

**Introduction**

This chapter covers the structure and contents of the log file into which the device enters the outcome of the respective commands.

**Contents**


| Topic | Page |
|---|---|
| File contents | 601 |

## File contents

**Introduction**

The log file is a plain text file. By making an entry into the command file, you define whether a log file is to be created or whether the device is to append the entries to an existing log file.

**Example**

```
JetControl AutoCopy log file 07.11.2008 09:14:09

 1: Ok      - FileCopy   /SD/OS/JC-340_1.04.0.00.os
                         /System/OS/op_system.os (345740 byte)
 2: Ok      - FileCopy   /SD/OS/JX3-AO4_1.01.0.00.os
                         /System/JX3-Module02/OS/system.os
                         (16832 byte)
 3: Warning - DirCreate  /app
 4: Ok      - DirCreate  /app/userprogtest
 5: Ok      - FileCopy   /SD/UserProgs/start.ini
                         /app/start.ini (63 byte)
 6: Ok      - FileCopy   /SD/UserProgs/userprogtest.es3
                         /app/userprogtest/userprogtest.es3
                         (169 byte)
 7: Error   - DaFileRead /SD/UserData/MyTestData.da

Command statistics:
  Total  : 7
  Ok     : 5
  Warning: 1
  Error  : 1
```

**Description**

When for each executed AutoCopy function a section is appended to an existing log file, the log file consists of three elements:

- The header contains date and time
- The following block contains information on the executed commands.
- Finally, it contains short statistics on command processing.

In the above example an error occurs when trying to create the directory */app* as this directory already exists. The device enters this error as a warning. When the device reads the DA file, an error also occurs. The device enters this error into the log file.

# 11.4 Data files

**Introduction**

This chapter covers data files where register and flag values are stored.

**Contents**


| Topic | Page |
|---|---|
| File format | 603 |

# File format

**Format**

The data file consists of the following elements:

- Pure text file
- Each entry must be in a separate line of text
- Each line must be terminated by carriage return/line feed
- Comment lines must be preceded by ";"
- Each data file is to start with the entry *SD1001*.

**Data lines**

A data line consists of the following elements:

- ID of the variable at the beginning of the line
- Now follows the number of the variable separated by a blank or tab
- Then follows the value of the variable separated by a blank or tab

| Variable ID | Variable type |
|---|---|
| FS | Flags |
| RS | Integer register |
| QA | Floating-point registers |

**Example**

```
SD1001
; Data File - Jetter AG
;
; Registers 1000000 ... 1000005
RS    1000000    12345
RS    1000001    2
RS    1000002    -1062729008
RS    1000003    502
RS    1000004    50
RS    1000005    3
QS    1009000    3.14
;
; Flags 10 ... 13
FS    10    0
FS    11    1
FS    12    1
FS    13    0
```

# 12 OS update

**Introduction**

Jetter AG are continuously striving to enhance the operating systems for their controllers and peripheral modules. Enhancing means adding new features, upgrading existing functions and fixing bugs.

This chapter describes how to perform an operating system update for a system equipped with a JC-360.

**Downloading an operating system**

You can download operating systems from the Jetter AG **homepage http://www.jetter.de**. You get the OS files for download at *Industrial Automation - Support - Downloads* or by clicking the quick link *Operating System Download* on the website of the corresponding controller or module.

**JC-3xx system - Devices**

The following devices within a system equipped with the JC-360 let you update the OS:

- Controller JC-360
- JX2 slave modules on a JX2 system bus
- Bus nodes on a JX2 system bus
- Analog modules on a JX3 system bus

**Contents**


| Topic | Page |
|---|---|
| Updating the operating system of the controller | 606 |
| OS update of a JX module | 611 |

# 12.1 Updating the operating system of the controller

**Introduction**

This chapter describes how to carry out an OS update of the JC-360. You have got several options to transfer the OS file to the controller:

- From within the programming tool JetSym
- Via FTP connection
- From an SD card
- From the application program

**Contents**


| Topic | Page |
|---|---|
| OS update by means of JetSym | 607 |
| Operating system update via FTP | 608 |
| Automatic OS update from an SD card | 609 |
| Operating system update from within the application program | 610 |

# OS update by means of JetSym

**Introduction**

The programming tool JetSym offers an easy way to transfer an OS file to the JC-360.

**Prerequisites**

- An OS file for the JC-360 must be available.
- A UDP/IP and a TCP/IP connection between programming tool and JC-360 is possible.
  The number of the IP port is set in the configuration memory as IP basic port number for JetIP communication.
- While booting, the controller must wait for the OS update, or the OS must already be running.
  **Note:**
  Make sure the controller remains energized.

**Updating the OS**

To update the OS, proceed as follows:

| Step | Action |
|---|---|
| 1 | Select in the JetSym menu **Build** the menu item **Update OS.** Alternative: In the **Advanced Configuration** dialog of the Hardware Manager, click on the button **Update OS**.<br>**Result:** The file selection dialog opens. |
| 2 | Select the new OS file here.<br>**Result:** In JetSym, a confirmation dialog opens. |
| 3 | Launch the OS upload by clicking the button **Yes**. |
| 4 | Wait until the update process is completed. |
| 5 | To activate the transferred OS, re-boot the controller. |

## Operating system update via FTP

**Introduction**

Using an FTP client an OS file can be transferred to the JC-360.

**Prerequisites**

- An OS file for the JC-360 must be available.
- An FTP connection to the controller must be possible.
- The login parameters for a user with administrator or system rights are at hand.
- The operating system of the JC-360 must be running.
  **Note:**
  Make sure it remains energized.

**Updating the OS**

To update the OS, proceed as follows:

| Step | Action |
|---|---|
| **1** | Open an FTP connection to the JC-360. |
| **2** | Log in with administrator or system rights |
| **3** | Navigate to the directory */System/OS*. |
| **4** | Transfer the OS file. |
| **5** | Wait until the transfer process is completed. |
| **6** | Clear the FTP connection. |
| **7** | To activate the transferred OS, re-boot the controller. |

## Automatic OS update from an SD card

**Reference:** An automatic OS update of the controller from the SD card can be carried out using the AutoCopy function. For a detailed description, turn to the chapter *AutoCopy* (see page 580).

# Operating system update from within the application program

**Introduction**

The file functions included in the STX language let you carry out a program-controlled OS update of a JC-360 from within an OS file.

**Prerequisites**

- An OS file must be available in the file system of the JC-360.
- The operating system of the JC-360 and the application program must be running.
  **Note:**
  Make sure it remains energized.

**Updating the OS**

To start an OS update out of the application program, proceed as follows:

| Step | Action |
|---|---|
| 1 | Copy the OS file to a file of any name and of the extension ***.os** in the directory */System/OS*. |
| 2 | To activate the transferred OS, for example by writing to the system command register, re-boot the controller. |

**JetSym STX program**

```
Var
    SourceName:             String;
    DestinationName:        String;
    UpdateIt:               Bool;
End_Var;

//*****************************************************
// Name:      OSupdate
// 1. Activate 'Tracing' in JetSym
// 2. Set the name of the source file in 'SourceName'
// 3. Set the flag 'UpdateIt'
//*****************************************************

Task OSupdate Autorun
    Var
        ResCopy:    Int;
    End_Var;

    DestinationName := '/System/OS/OperatingSystem.os';
    Loop
        UpdateIt := False;
        When UpdateIt Continue;
        ResCopy := FileCopy(SourceName,
                            DestinationName);
        Trace('Result : ' + IntToStr(ResCopy) + '$n');
    End_Loop;
End_Task;
```

# 12.2 OS update of a JX module

**Introduction**

This chapter describes how to carry out an OS update of a JX3 module on a JX2 or JX3 system bus which is connected to the JC-360. You have got several options to transfer the OS file to the JX module:

- From within the programming tool JetSym
- Via FTP connection
- From an SD card
- From the application program

**Contents**


| Topic | Page |
|---|---|
| OS update by means of JetSym | 612 |
| Operating system update via FTP | 613 |
| Automatic OS update from an SD card | 614 |
| Operating system update from within the application program | 615 |

# OS update by means of JetSym

**Introduction**

The programming tool JetSym offers an easy way to transfer an OS file to a JX module connected to the JX2 or JX3 system bus of the controller.

**Prerequisites**

- An OS file for the JX module is available.
- A UDP/IP and a TCP/IP connection between programming tool and controller is possible.
  The number of the IP port is set in the configuration memory as IP basic port number for JetIP communication.
- The operating system is running.
- The controller has initialized both the JX2 and JX3 system bus with the connected JX modules without errors.
  **Note:**
  Make sure the controller and the JX modules remain energized.

**Updating the OS**

To update the OS of a JX module, proceed as follows:

| Step | Action | |
|---|---|---|
| 1 | Select in the JetSym menu **Build** the menu item **Update OS.** Alternative: In the **Advanced Configuration** dialog of the Hardware Manager, click on the button **Update OS**.<br>**Result:** A file selection dialog opens. | |
| 2 | Select the new OS file here.<br>**Ergebnis:** In JetSym, a confirmation dialog opens. | |
| 3 | Confirm by clicking **Yes.**<br>**Ergebnis:** JetSym opens an input box for entering the interface type and module number. | |
| 4 | Enter here the interface type (1 for the JX3 system bus; 2 for the JX2 system bus) and the module number (2 ... 23).<br>Launch the OS upload by clicking the button **Update**. | |
| 5 | Wait until the update process is completed. | |
| 6 | **If ...** | **... then ...** |
| | ... you wish to update further JX3 modules, ... | proceed with step 1 |
| | ... you do not wish to update any other JX3 modules, ... | ... reboot the controller to launch the new operating system. |

# Operating system update via FTP

**Introduction**

Using an FTP client an OS file can be transferred to a JX module connected to the JX2 or JX3 system bus of the controller.

**Prerequisites**

- An OS file for the JX module is available.
- An FTP connection to the controller must be possible.
- The login parameters for a user with administrator or system rights are at hand.
- The operating system of the controller must be running.
- The controller has initialized both the JX2 and JX3 system bus with the connected JX modules without errors.

  **Note:**

  Make sure the controller and the JX modules remain energized.

**Updating the OS**

To update the OS of a JX module, proceed as follows:

| Step | Action | |
|---|---|---|
| **1** | Establish an FTP connection to the controller | |
| **2** | Log in with administrator or system rights | |
| **3** | Navigate to the OS directory of the JX module.<br>Example: */System/JX2-Slave02/OS* or */System/JX3-Module05/OS* | |
| **4** | Transfer the OS file into this directory. | |
| **5** | Wait until the update process is completed. | |
| **6** | **If ...** | **... then ...** |
| | ... you wish to update further JX3 modules, ... | proceed with step 3 |
| | ... you do not wish to update any other JX3 modules, ... | ... clear the FTP connection. Then reboot the controller to launch the new operating system. |

## Automatic OS update from an SD card

**Reference:**

An automatic OS update of a JX2 or JX3 module from the SD card can be carried out using the AutoCopy function. For a detailed description, turn to the chapter *AutoCopy* (see page 580).

# Operating system update from within the application program

**Introduction**

The file functions included in the STX language scope let you transfer an OS ile to a JX module on the JX2 or JX3 system bus of the controller.

**Prerequisites**

- An OS file for the JX module is available in the file system of the controller.
- The operating system of the controller and the application program must be running.
- The controller has initialized both the JX2 and JX3 system bus with the connected JX modules without errors.
  **Note:**
  Make sure the controller and the JX modules remain energized.

**Updating the OS**

To update the OS of a JX module, proceed as follows:

| Step | Action | |
|---|---|---|
| 1 | Copy the OS file to a file of any name and of the extension ***.os** in the OS file of the module.<br>Example: */System/JX2-Slave02/OS/system.os* or */System/JX3-Module05/OS/Anything.os* | |
| 2 | **If ...** | **... then ...** |
| | ... you wish to update further JX3 modules, ... | ... proceed with step 1 |
| | ... you do not wish to update any other JX3 modules, ... | ... reboot the controller to launch the new operating system. |

**JetSym STX program**

```
Var
    SourceName:             String[100];
    DestinationName:        String[100];
    UpdateIt:               Bool;
End_Var;
```

```
//*****************************************************
// 1. Activate 'Tracing' in JetSym
// 2. Set the name of the source file in 'SourceName'
// 3. Set the name of the target file in 'DestinationName'
// 4. Set the flag 'UpdateIt'
//*****************************************************
Task OSupdate Autorun
    Var
        ResCopy:    Int;
    End_Var;

    Loop
        UpdateIt := False;
        When UpdateIt Continue;
        ResCopy := FileCopy(SourceName,
                            DestinationName);
        Trace('Result : ' + IntToStr(ResCopy) + '$n');
    End_Loop;
End_Task;
```

# 13 Application program

**Introduction**

This chapter describes how to store the application program in JC-360. The user determines the program that is to be executed.

**Required programmer's skills**

This chapter requires knowledge on how to create application programs in JetSym and how to transmit them via the file system of the JC-360.

**Contents**


| Topic | Page |
|---|---|
| Application program - Default path | 618 |
| The application program is stored to the SD card | 619 |
| Loading an application program | 620 |

# Application program - Default path

**Introduction**

When uploading the application program from JetSym to the controller, this program is stored as a file to the internal flash disk. The device enters the path and file name into the file **/app/start.ini**.

**Path and file name**

In the directory */app*, JetSym, by default, creates a subdirectory and assigns the project name to it. Then, JetSym stores the application program to this subdirectory assigning the extension ***.es3** to it. The path and file names are always converted into lower case letters.

**/app/start.ini - Structure**

This file is a text file with one section holding two entries:

| Element | Description |
|---|---|
| *[Startup]* | Section name |
| *Project* | Path to the application program file. This path is relative to */app*. |
| *Program* | Name of the application program file |

**Example:**

```
[Startup]
Project = test_program
Program = test_program.es3
```

**Result:**The application program is loaded from the file **/app/test_program/test_program.es3**.

**Related topics**

- **Storing the application program to the SD card** (see page 619)

# The application program is stored to the SD card

**Introduction**

When uploading the application program from JetSym to the controller, the default storage for application programs is used.

If you want the device to read the application program from the SD card, you have to configure the file path.

If you want to store the application program to another directory of the internal flash disk, proceed the same way.

**Prerequisites**

Since the controller's file system is case sensitive, make sure that path and file names, as well as file entries are spelled correctly.

**Storing the application program to the SD card**

To store the application program to the SD card, configure the device as follows:

| Step | Action |
|---|---|
| **1** | Create an application program file by JetSym. |
| **2** | Create the desired directory on the SD card. |
| **3** | Store the application program file to the desired directory. |
| **4** | Enter the path to the application program file and the program name into the file **/app/start.ini** on the internal flash disk of the device. |

**Result:** On re-boot, the device loads the application program from the SD card.

**/app/start.ini - Structure**

This file is a text file with one section holding two entries:

| Element | Description |
|---|---|
| [Startup] | Section name |
| Project | Path leading to the application program file |
| Program | Name of the application program file |

**Example:**

```
[Startup]
Project = /SD/TestProgram
Program = Test1.es3
```

**Result:** The application program is loaded from the file **Test1.es3** located in the folder **TestProgram** on the SD card (**/SD/TestProgram/Test1.es3**).

**Related topics**

- **Application program - Default path** (see page 618)

## Loading an application program

**Introduction**

At reboot of the application program via JetSym or booting the JC-360, the application program is loaded and executed via the file system. For this, mode selector S11 must be in *RUN* position.

**The loading process**

The application program is loaded by the controller's OS as follows:

| Step | Description |
|---|---|
| **1** | The OS reads the file **/app/start.ini** from the internal flash disk. |
| **2** | The OS evaluates the **Project** entry. It contains the path leading to the application program file. |
| **3** | The OS evaluates the **Program** entry. It contains the program name. |
| **4** | The OS loads the application program from the file **<Project>/<Program>**. |

# 14 Quick reference - JC-3xx

## Corresponding OS version

This quick reference gives a compact overview of registers and flags used in connection with the controllers JC-340 andJC-350 with OS release 1.24.0.00, as well as JC-360(MC) and JC-365(MC) with OS release 1.26.0.00.
Further, the allocation of plug-in connectors and setting the IP address via DIP switch have been described.

## Default address on the CANopen® bus

The default address of the supplied JC-360:
**Node-ID: 127 (0x7F)**

## Maximum number of CANopen® interfaces

**JC-340 and JC-350:**
Possible number of CAN interfaces (CANopen®-STX-API): 0

**JC-360(MC):**
Possible number of CAN interfaces (CANopen®-STX-API): 1
**CANMAX: 0**

**JC-365(MC):**
Possible number of CAN interfaces (CANopen®-STX-API): 2
**CANMAX: 1**

## Heartbeat monitoring (CANopen®-STX-API) available as of the following OS version

**JC-340 and JC-350:**
CANopen-STX-API is not available.

**JC-360(MC):**
OS version: 1.17.0.25

**JC-365(MC):**
OS version: 1.17.0.25

## General overview - Registers

| | |
|---|---|
| 100000 ... 100999 | Electronic Data Sheet (EDS) |
| 101000 … 101999 | Configuration |
| 102000 … 102999 | Real-time clock |
| 103000 … 103999 | Serial interface |
| 104000 ... 104999 | Ethernet |
| 107000 ... 107499 | SD card |
| 107500 ... 107599 | Flash disk |
| 108000 ... 108999 | CPU/backplane |
| 200000 ... 209999 | General system registers |
| 210000 ... 219999 | Application program |
| 220000 ... 229999 | HMI control |
| 230000 ... 239999 | Networking via JetIP |
| 240000 ... 249999 | JetSync |
| 250000 ... 259999 | Ethernet system bus |
| 260000 ... 269999 | RemoteScan |
| 270000 ... 279999 | Modbus/TCP |
| 290000 ... 299999 | E-mail |
| 310000 ... 319999 | File system/data files |
| 320000 ... 324999 | FTP client |
| 350000 ... 359999 | User-programmable IP interface |
| 380000 ... 389999 | Error history |
| 390000 ... 399999 | I/O networking |
| 470000 ... 479999 | NetConsistency |
| 510000 ... 519999 | DNS server/DNS cache |
| 520000 ... 529999 | JetIPScan |
| 1000000 ... 1001999 | JC-340: Application registers (remanent; integer/float) |
| 1000000 ... 1019999 | JC-340: Application registers (remanent; integer/float) with option -SD |
| 1000000 ... 1029999 | JC-350: Application registers (remanent; integer/float) |
| 1000000 ... 1059999 | JC-360/JC-365: Application registers (remanent; integer/float) |
| 1000000 ... 1119999 | JC-360/JC-365: Application registers (remanent; integer/float) with option -R |
| 100xx0000 ... 100xx9999 | JX3 modules (xx: 02 ... 17); |
| 200002000 ... 200029999 | JX2 system bus |
| | Networking via Jetter Ethernet system bus GNN: nnn = 000 ... 199 |
| 1nnn020000 ... 1nnn179999 | JX3 module registers |
| 1nnn202000 ... 1nnn227999 | JX2 module registers |
| 1nnn810000 ... 1nnn819999 | JetMove registers |
| 1nnn980000 ... 1nnn980199 | Indirect access via local R 236xxx |
| 1nnn990000 ... 1nnn999999 | Indirect access with variable target window |

## I/Os - General overview

| | |
|---|---|
| 20001 ... 36000 | Virtual I/Os for RemoteScan |
| 10000xx01 ... 10000xx16 | JX3 modules (xx: 02 ... 17) |
| 20000xx01 ... 20000xx16 | JX2 modules (xx: 02 ... 24) |
| 1nnn010101 ... 1nnn011716 | JX3 modules via JX3-BN-ETH GNN: nnn = 000 ... 199 |

## Flags - General overview

| | |
|---|---|
| 0 ... 255 | Application flags (remanent) |
| 256 ... 2047 | Overlaid by registers R 1000000 through 1000055 |
| 2048 ... 2303 | Special flags |

## Electronic Data Sheet (EDS)

| | |
|---|---|
| 100500 | Interface (0 = CPU, 1 = JX3 modules) |
| 100501 | Module number (2 ... 17)<br>If <100500> = 0:<br>The EDS of the controller is displayed.<br>If <100500> = 1 and <100501> = 2 ... 17:<br>The EDS of the selected JX3 module is displayed. |
| **[Identification]** | |
| 100600 | Internal version number |
| 100601 | Module ID |
| 100602 ... 100612 | Module name (register string) |
| 100613 | PCB revision |
| 100614 | PCB options |
| **[Production]** | |
| 100700 | Internal version number |
| 100701 ... 100707 | Serial number (register string) |

| | |
|---|---|
| 100708 | Day |
| 100709 | Month |
| 100710 | Year |
| 100711 | TestNum. |
| 100712 | TestRev. |
| **[Features]** | **I/O module** |
| 100800 | Internal version number |
| 100801 | Diagnostic configuration |
| 100802 | Digital inputs |
| 100803 | Digital inputs, inverted |
| 100804 | Digital outputs |
| 100805 | Digital outputs, inverted |
| 100806 | Cyclic inputs |
| 100807 | Cyclic outputs |
| 100808 | Features |
| 100809 | Diagnostics mask |
| **[Features]** | **JX3-BN-ETH/JC-3xx** |
| 100800 | Internal version number |
| 100801 | MAC address (Jetter) |
| 100802 | MAC address (device) |
| 100803 | Serial port |
| 100804 | Switch |
| 100805 | STX |
| 100806 | Non-volatile registers |
| 100807 | JX3 bus |
| 100808 | CAN bus |
| 100809 | SD card |
| 100810 | Motion control |
| 100811 | Intelligent slave modules |
| 100812 | HTTP/e-mail |
| 100813 | Modbus/TCP |
| 100815 | LED for the SD card |
| 100816 | User-defined LEDs |
| 100817 | RTC |

## Configuration

| | |
|---|---|
| | **From the file /system/config.ini** |
| 101100 | IP address |
| 101101 | Subnet mask |
| 101102 | Default gateway |
| 101103 | DNS server |
| 101132 | HOSTNAME suffix type |
| 101133 ... 101151 | HOSTNAME (register string) |
| 101164 | Port number for JetIP |
| 101165 | Port number for STX debugger |
| | **Used by the system** |
| 101200 | IP address |
| 101201 | Subnet mask |
| 101202 | Default gateway |
| 101203 | DNS server |
| 101232 | HOSTNAME suffix type |
| 101233 … 101251 | HOSTNAME (register string) |
| 101264 | Port number for JetIP |
| 101265 | Port number for STX debugger |
| 101280 ... 101298 | File name for AutoCopy |
| 101299 | Saving the settings (0x77566152) |
| 101908 | CRC of ModConfig.da |

## Real-time clock

| | |
|---|---|
| | **Direct access** |
| 102910 | Milliseconds |
| 102911 | Seconds |
| 102912 | Minutes |
| 102913 | Hours |
| 102914 | Weekday (0 = Sunday) |
| 102915 | Day |
| 102916 | Month |
| 102917 | Year |
| | **Buffer access** |
| 102920 | Milliseconds |
| 102921 | Seconds |
| 102922 | Minutes |
| 102923 | Hours |
| 102924 | Day of the week (0 = Sunday) |
| 102925 | Day |
| 102926 | Month |
| 102927 | Year |
| 102928 | Read/write trigger |

## Serial interface

| | |
|---|---|
| 103000 | Error state (bit-coded)<br>Bit 14 = 1: Framing error<br>Bit 13 = 1: Parity error<br>Bit 12 = 1: Overflow |
| 103001 | Protocol<br>1: System logger<br>2: Prim<br>3: pcomX |
| 103002 | Baud rate (1,200 … 115,200) |
| 103003 | Bits per character (5 ... 8) |
| 103004 | Stop bits (1, 2) |
| 103005 | Parity<br>0: None<br>1: Odd<br>2: Even<br>3: 1<br>4: 0 |
| 103006 | 0 = RS-232, 1 = RS-422, 3 = RS-485/2 |
| 103010 | Transmit buffer |
| 103011 | Transmit buffer filling level |
| 103012 | Receiving buffer (without immediate clearing) |
| 103013 | Receiving buffer (with immediate clearing) |
| 103014 | Receiving buffer filling level |
| 103015 | Receiving buffer, 16-bit, little endian |
| 103016 | Receiving buffer; 16-bit; big endian |
| 103017 | Receiving buffer, 32-bit, little endian |
| 103018 | Receiving buffer; 32-bit; big endian |
| 103019 | Error counter |

## Ethernet

| | |
|---|---|
| | **Ethernet** |

| | |
|---|---|
| 104100 ... 104156 | MIB counter |
| | **ARP** |
| 104200 | Transmitted requests |
| 104201 | Received requests |
| 104202 | Transmitted responses |
| 104203 | Received responses |
| 104204 | Dynamic entries |
| 104205 | Static entries |
| 104206 | Obsolete entries |
| 104250 | Executing an ARP request |
| 104350 | GNN |
| | **IP** |
| 104500 | Transmitted packets |
| 104501 | Transmitted bytes |
| 104502 | Received packets |
| 104503 | Received bytes |
| 104504 | Invalid packets |
| 104505 | Discarded received packets |
| 104506 | Checksum error at reception |
| 104507 | Discarded transmitted packets |
| 104508 | Transmitted fragments |
| 104509 | Received fragments |
| 104531 | Current IP address (rw) |
| 104532 | Current subnet mask (rw) |
| 104533 | Current default gateway (rw) |
| 104534 | IP address of DNS server (rw) |
| | **TCP** |
| 104800 | Transmitted packets |
| 104801 | Transmitted bytes |
| 104802 | Received packets |
| 104803 | Received bytes |
| 104804 | Invalid packets |
| 104805 | Discarded received packets |
| 104806 | Checksum error |
| 104807 | Connections |
| 104808 | Disconnections |
| 104809 | Discarded connections |
| 104810 | Repeated transmitted packets |
| | **UDP** |
| 104900 | Transmitted packets |
| 104901 | Transmitted bytes |
| 104902 | Received packets |
| 104903 | Received bytes |
| 104904 | Invalid packets |
| 104905 | Discarded received packets |
| 104906 | Checksum error |

## SD card

| | |
|---|---|
| 107000 | Bit 0 = 1: Card installed |
| | Bit 1 = 1: Card is ready |
| 107001 | 1 = Card is write-protected (only valid if R 107000 = 3) |
| 107002 | Size in MB |

## Flash disk

| | |
|---|---|
| 107500 | Status |
| 107501 | Command<br>30: Read statistics<br>**Sector statistics** |
| 107510 | Total |
| 107511 | Used |
| 107512 | Blocked |
| 107513 | Unassigned |
| | **Byte statistics** |
| 107520 | Total |
| 107521 | Used |
| 107522 | Blocked |
| 107523 | Unassigned |

## CPU/backplane module

| | |
|---|---|
| 108002 | All LEDs on/off (bit-coded) |
| | Bit 0: LED **R** |
| | Bit 1: LED **E** |
| | Bit 2: LED **D1** |
| | Bit 3: LED **D2** |
| 108003 | LED **R**<br>0 = OFF |
| | 1 = Flashing slowly |
| | 2 = Flashing fast |
| | 3 = ON |
| 108004 | LED **E**<br>0 = OFF |
| | 1 = Flashing slowly |
| | 2 = Flashing fast |
| | 3 = ON |
| 108005 | LED **D1**<br>0 = OFF |
| | 1 = Flashing slowly |
| | 2 = Flashing fast |
| | 3 = ON |
| 108006 | LED **D2**<br>0 = OFF |
| | 1 = Flashing slowly |
| | 2 = Flashing fast |
| | 3 = ON |
| 108007 | LED **SD** |
| | 0 = OFF |
| | 3 = ON |
| 108008 | LEDs **U1** through **U4** on/off (bit-coded) |
| | Bit 0: LED **U1** |
| | Bit 1: LED **U2** |
| | Bit 2: LED **U3** |
| | Bit 3: LED **U4** |
| 108010 | DIP switch - All switches |
| 108011 | DIP switch - Address |
| 108012 | DIP switch - Mode |

| | |
|---|---|
| 108015 | Mode selector |
| | 1 = LOAD |
| | 2 = RUN |
| | 3 = STOP |
| 108020 | Revision of the backplane module |
| 108021 | CPU board revision |
| 108099 | Clear EEPROM (0x12345678) |
| 108100 ... 108227 | EEPROM registers on the backplane module |

## General system registers

| | |
|---|---|
| 200000 | OS version (major * 100 + minor) |
| 200001 | Application program is running (bit 0 = 1) |
| | 0/2: Stop program |
| | 1: Start program |
| | 3: Continue program |
| 200008 | Error register 1 (identical with 210004) |
| | Bit 0: Error on flash disk |
| | Bit 1: Error on JX3 system bus |
| | Bit 2: Error on JX2 system bus |
| | Bit 3: Error on Ethernet system bus |
| | Bit 7: Error in expanded error register |
| | Bit 8: Illegal jump |
| | Bit 9: Illegal call |
| | Bit 10: Illegal index |
| | Bit 11: Illegal opcode |
| | Bit 12: Division by 0 |
| | Bit 13: Stack overflow |
| | Bit 14: Stack underflow |
| | Bit 15: Illegal stack |
| | Bit 16: Error when loading the application program |
| | Bit 17: Memory protection violated |
| | Bit 24: Timeout - Cycle time |
| | Bit 25: Timeout - Task lock |
| | Bit 31: Unknown error |
| 200009 | Enhanced error register 1 (bit-coded) |
| | Bit 3: Error in ModConfig.da |
| | Bit 10: A bus node (publish/subscribe client) has reported an error |
| | Bit 12: JetIPScan has reported an error |
| | Bit 16: NetConsistency has reported an error |
| | Bit 24: Applies to JC-360(MC) and JC-365(MC) only: IP address conflict detected |
| 200010 | Enhanced error register 2 (bit-coded) |
| | Bit 1: Error in the MC object |
| 200051 | Error numbers of JetIPScan |
| | 0: No error or warning |
| | 5: The user has terminated the function |
| | 1001: The first received response does not match response 2 and 3 |
| | 1002: The second received response does not match response 1 and 3 |
| | 1003: The third received response does not match response 2 and 3 |
| | -1: All three responses are dissimilar |
| | -2: The IP settings of at least one node are dissimilar. |
| | -3: The JetIPScan function has been invoked, although it is active already |
| | -10: The length of the set value list is < 1 or > 255, or the pointer to the list is invalid |
| | -11: A GNN of the set value list is < 1 or > 255, or it is a multiple GNN |
| | -20 ... -40: Internal error |
| | -1001 ... -1199: The node has reported the wrong CtrlID or CtrlIDopt |
| | -2001 ... -2199: The node has not called |
| | -3001 ... -3199: Several nodes of the same GNN have called |
| 200061 | Error numbers of NetConsistency, see R 470040 |
| 200010 | Enhanced error register 2 (bit-coded) |
| | Bit 1: Error in the MC object |
| 200168 | Bootloader version (IP format) |
| 200169 | OS version (IP format) |
| 200170 | Controller type (340/350/360) |
| 200300 | Currently available heap |
| 200301 | Available heap at system launch |
| 200302 | Available heap before application program |
| 201000 | Runtime register in milliseconds (rw) |
| 201001 | Runtime register in seconds (rw) |
| 201002 | Runtime register in R 201003 |
| | Units (rw) |
| 201003 | * 10 ms units for R 201002 (rw) |
| 201004 | Runtime register in milliseconds (ro) |
| 201005 | Runtime registers in microseconds (ro) |
| 202930 | Web status (bit-coded) |
| | Bit 0 = 1: FTP server available |
| | Bit 1 = 1: HTTP server available |
| | Bit 2 = 1: E-mail available |
| | Bit 3 = 1: Data file function available |
| | Bit 4 = 1: Modbus/TCP has been licensed |
| | Bit 5 = 1: Modbus/TCP available |
| | Bit 6: Reserved |
| | Bit 7 = 1: FTP client available |
| 202936 | Control register of the file system |
| | 0xc4697a4b: Formatting the flash disk |
| | 0xd364e64d: Formatting the SD card |
| | 0x2c9b3c94: Checking the SD card |
| 202960 | Password for system command register (0x424f6f74) |
| 202961 | System command register |
| | 102: Controller restart (reboot) |
| | 104: Reset remanent parameters |
| | 122: Wait for communication - OFF |
| | 123: Wait for communication - ON |
| | 160: Task switch on I/O access - OFF |
| | 161: Task switch on I/O access - ON |
| | 170: Continue task time slice - OFF |

| | |
|---|---|
| | 171: Continue task time slice - ON |
| | 310: Load the configuration data |
| | 311: Load the module configuration |
| | 312: Load process data configuration for Ethernet system bus |
| | 313: Stop Ethernet system bus process data configuration |
| | 330: JetIPScan client - OFF |
| | 331: JetIPScan client - ON |
| | 410: JetSync blockage - OFF |
| | 411: JetSync blockage for all ports - ON |
| | 412: JetSync blockage for port X15 - ON |
| 202962 | System status register |
| | Bit 0 = 1: Task switch on I/O access |
| | Bit 1 = 1: Without waiting for communication |
| | Bit 2 = 1: JetIPScan client - ON |
| | Bit 3 = 1: Continue task time slice - ON |
| | Bit 8 = 1: JetSync blockage - ON |
| 202970 | Password for start delay (0x424f6f74) |
| 202971 | Start delay in steps of 100 ms |
| 203000 | Interface monitoring: JetIP |
| 203001 | Interface monitoring: SER |
| 203005 | Interface monitoring: STX debug server |
| 203100 ... 203107 | 32-bit overlaying - Flag 0 ... 255 |
| 203108 ... 203123 | 16-bit overlaying - Flag 0 ... 255 |
| 203124 ... 203131 | 32-bit overlaying - Flag 2048 ... 2303 |
| 203132 ... 203147 | 16-bit overlaying - Flag 2048 ... 2303 |
| 209700 | System logger: Global enable |
| 209701 ... 209739 | Enabling system components |

## Application program

| | |
|---|---|
| 210000 | Application program is running (bit 0 = 1) |
| | 0/2: Stop program |
| | 1: Start program |
| | 2: Continue program |
| 210001 | JetVM version |
| 210004 | Error register (bit-coded) |
| | Bit 1: Error on JX3 system bus |
| | Bit 2: Error on JX2 system bus |
| | Bit 3: Error on Ethernet system bus |
| | Bit 7: Error in expanded error register |
| | Bit 8: Illegal jump |
| | Bit 9: Illegal call |
| | Bit 10: Illegal index |
| | Bit 11: Illegal opcode |
| | Bit 12: Division by 0 |
| | Bit 13: Stack overflow |
| | Bit 14: Stack underflow |
| | Bit 15: Illegal stack |
| | Bit 16: Error when loading the application program |
| | Bit 24: Timeout - Cycle time |
| | Bit 25: Timeout - Task lock |
| | Bit 31: Unknown error |
| 210006 | Highest task number |
| 210007 | Minimum program cycle time |
| 210008 | Maximum program cycle time |
| 210009 | Current program cycle time |
| 210011 | Current task number |
| 210050 | Current program position within an execution unit |
| 210051 | ID of the execution unit being processed |
| 210056 | Desired total cycle time in µs |
| 210057 | Calculated total cycle time in µs |
| 210058 | Maximum time slice per task in µs |
| 210060 | Task ID (for R210061) |
| 210061 | Priority for task [R210060] |
| 210063 | Length of scheduler table |
| 210064 | Index in scheduler table |
| 210065 | Task ID in scheduler table |
| 210070 | Task ID (for R210071) |
| 210071 | Timer number (0 ... 31) |
| 210072 | Manual triggering of a timer event (bit-coded) |
| 210073 | End of cyclic task (task ID) |
| 210074 | Command for cyclic tasks |
| 210075 | Number of timers |
| 210076 | Timer number (for R210077) |
| 210077 | Timer value in milliseconds |
| 210091 | Debugging - STX variable address |
| 210093 | Debugging - STX variable value |
| 210100 ... 210199 | Task state<br>Apply the STX function TaskGetInfo() as described in the JetSym online help. |
| 210400 ... 210499 | Task - Program address |
| 210600 | Task ID of a cyclical task (for R210601) |
| 210601 | Processing time of a cyclical task in per mil figure |
| 210609 | Task lock timeout in ms |
| | -1: Monitoring disabled |
| 210610 | Timeout (bit-coded, |
| | bit 0 -> timer 0, etc.) |
| | **TCP auto-close for the STX debug server** |
| 212000 | Number of open connections |
| 212001 | Mode |
| 212002 | Time |

## HMI control

| | |
|---|---|
| 222804 | Total number of display characters |
| 222805 | Number of characters per line |
| 222806 | Text selection (DisplayText2) |
| 222808 | Number of decimal places (UserInput) |
| 222810 | Number of decimal places (DisplayValue) |
| 222811 | Max. number of decimal places (UserInput) |
| 222812 | Field length (DisplayValue) |

| | |
|---|---|
| 222813 | Field length (UserInput) |
| 222814 | Indirect cursor position |
| 222815 | Default value for UserInput (Integer/Float) |
| 222816 | Displaying the sign |
| 222817 | Status of UserInput |
| 222818 | Enable/disable monitor functions |
| 222819 | Display text - monitor function |
| 222820 | Switching over to monitor display |
| 222821 | Dialog language |
| 222824 | Indirect buffer number |
| | **Multi-display mode** |
| 222825 | Text buffer for display 1 |
| 222826 | Text buffer for display 2 |
| 222827 | Text buffer for display 3 |
| 222828 | Text buffer for display 4 |
| 222829 | Basic flag number for display 1 |
| 222830 | Basic flag number for display 2 |
| 222831 | Basic flag number for display 3 |
| 222832 | Basic flag number for display 4 |
| 222833 | Register number - LED display 1 |
| 222834 | Register number - LED display 2 |
| 222835 | Register number - LED display 3 |
| 222836 | Register number - LED display 4 |
| 222837 | Module number of PRN (display redirection) |
| 222838 | Module number of SER (display redirection) |
| 222839 | Character code for *Delete display* |
| 222840 | Character code for *Delete to end of line* |

## Networking via JetIP

| | |
|---|---|
| | **TCP auto-close for the JetIP/TCP server** |
| 230000 | Number of open connections |
| 230001 | Mode |
| 230002 | Time |
| | **Other registers for networking via JetIP** |
| 232708 | Timeout in milliseconds |
| 232709 | Response time in milliseconds |
| 232710 | Amount of network errors |
| 232711 | Error code of last access<br>0 = No error |
| | 1 = Timeout |
| | 3 = Error message from the remote station |
| | 5 = Invalid network address |
| | 6 = Invalid amount of registers |
| | 7 = Invalid interface number |
| 232717 | Max. number of retries |
| 232718 | Number of retries |
| | **Network registers** |
| 235000 ... 235399 | IP addresses |
| 235400 ... 235799 | Port numbers |
| 236000 ... 236399 | Indirect register numbers |
| | GNN: nnn = 000 ... 199 |
| 1nnn020000 ... 1nnn179999 | JX3 module registers |
| 1nnn202000 ... 1nnn227999 | JX2 module registers |
| 1nnn810000 ... 1nnn819999 | JetMove registers |
| 1nnn980000 ... 1nnn980199 | Indirect access via local register 236xxx |
| 1nnn990000 ... 1nnn999999 | Indirect access with variable target window |

## Ethernet system bus

| | |
|---|---|
| | **Subscriber** |
| 250000 | Status (bit-coded)<br>Bit 0 = 1: No CRC |
| | Bit 1 = 1: Error in connection with a subscription |
| | Bit 7 = 1: Subscriber is running |
| 250001 | Command<br>102: Restart |
| | 105: STOPP |
| | 110: Acknowledge error |
| 250002 | Subscription ID of the last error |
| 250003 | Number of subscriptions |
| 250004 | CRC of configuration file |
| 250010 | Selection via command |
| 250011 | Selection via ID |
| | **Subscription** |
| 250020 | Status |
| 250021 | Mode |
| 250022 | Number of elements |
| 250023 | Multicast group |
| 250024 | Hash |
| 250025 | Current sequence number |
| 250026 | Size (bytes) |
| 250027 | Timeout |
| 250028 | Number of received publications |
| 250029 | Number of timeout errors |
| 250030 | Amount of sequence number errors |
| 250100 ... 250999 | 9 more subscriber register blocks |
| | **Address of the bus node (or controller) exceeding the timeout time** |
| 254001 | GNN |
| 254002 | IP address |
| 254003 | Port number |
| | **Publisher** |
| 255000 | Status (bit-coded)<br>Bit 0 = 1: No CRC |
| | Bit 1 = 1: Error in connection with a publication |
| | Bit 7 = 1: Subscriber is running |
| 255001 | Command<br>102: Restart |
| | 105: STOPP |
| | 110: Acknowledge error |
| 255002 | Publication ID of the last error |
| 255003 | Number of publications |
| 255004 | CRC of configuration file |
| 255010 | Selection via command |
| 255011 | Selection via ID |
| | **Publication** |
| 255020 | Status |
| 255021 | Mode |
| 255022 | Number of elements |
| 255023 | Multicast group |
| 255024 | Hash |
| 255025 | Current sequence number |

| | |
|---|---|
| 255026 | Size (bytes) |
| 255027 | Cycle time |
| 255028 | Number of publications sent |
| 255029 | Number of retries |
| 255030 | Number of transmit errors |
| 255100 ... 255999 | 9 more publisher register blocks |

## RemoteScan

| | |
|---|---|
| 262965 | Protocol type |
| 262966 | Amount of configuration blocks |
| 262967 | Status |

## Modbus/TCP

| | |
|---|---|
| 272702 | Register offset |
| 272704 | Input offset |
| 272705 | Output offset |
| 278000 ... 278999 | 16-bit I/O registers overlaid by virtual I/Os 20001 ... 36000 |

## E-mail

| | |
|---|---|
| 292932 | IP address of the SMTP server |
| 292933 | IP address of the POP3 server |
| 292934 | Port number of the SMTP server |
| 292935 | Port number of POP3 server |
| 292937 | Status of e-mail processing |
| 292938 | Task ID - E-mail |

## File system/data file function

| | |
|---|---|
| 312977 | Status of file operation |
| 312978 | Task ID |

## FTP client

| | |
|---|---|
| 320000 | Number of open connections |
| 320001 | Command |
| 320002 | Timeout |
| 320003 | Server port |
| 320004 | Selection via number |
| 320005 | Selection via handle |
| 320006 | Server socket: IP address |
| 320007 | Server socket: Port |
| 320008 | Client socket: IP address |
| 320009 | Client socket: Port |
| 320100 | Access status |
| 320101 | Task ID |

## User-programmable IP interface

**Reading out the connection list**

| | |
|---|---|
| 350000 | Last result (-1 = no connection selected) |
| 350001 | 1 = Client; 2 = Server |
| 350002 | 1 = UDP; 2 = TCP |
| 350003 | IP address |
| 350004 | Port number |
| 350005 | Connection state |
| 350006 | Number of sent bytes |
| 350007 | Number of received bytes |

## Error history

| | |
|---|---|
| 380000 | Status<br>Bit 0 = 1: Recording<br>Bit 1 = 1: Stop if buffer is full<br>Bit 2 = 1: Stop on error code<br>Bit 3 = 1: Remanent memory |
| 380001 | Command<br>1: Clear error log<br>2: Start error log<br>3: Stop error log<br>4: Stop if error buffer is full<br>5: Circular buffer<br>6: Stop on error code ON<br>7: Stop on error code OFF<br>10: Remanent memory<br>11: Dynamic memory |
| 380002 | Buffer length |
| 380003 | Maximum buffer length |
| 380004 | Number of error entries |
| 380005 | Index to error list |
| 380006 | Error entry |
| 380007 | Error stop code |
| 380008 | Number of codes until stop |
| 380029 | Group index to error list |
| 380030 ... 380093 | 64 error entries |

## I/O networking

**Status registers**

| | |
|---|---|
| 390000 + node * 10 | Error register |
| 390001 + node * 10 | Enhanced error register 1 |
| 390002 + node * 10 | Enhanced error register 2 |
| 390003 + node * 10 | JetSync status |
| 390004 + node * 10 | Subscriber status |
| 390005 + node * 10 | Subscription ID of the last error |

**Address of a bus node (not of a controller) having reported an error**

| | |
|---|---|
| 394001 | GNN |
| 394002 | IP address |
| 394003 | Port number |

**Control register**

| | |
|---|---|
| 395000 + node * 10 | Command |

## NetConsistency function

Only for JC-340, JC-350, JC-940MC and JC-945MC.

**Basic drivers**

| | |
|---|---|
| 470000 ... 470008 | Cookie |
| 470009 | Version |
| 470010 | Status<br>Bit 0 = 1: Error<br>Bit 1 = 1: Alarms<br>Bit 2 = 1: Basic driver initialized |
| 470011 | Command<br>0: There are no commands |
| 470020 | Maximum possible number of instances |
| 470021 | Number of instances ready for operation |
| 470030 | Max. number of error messages for the logger |
| 470031 | Number of error messages transmitted to the logger |
| 470032 | Max. number of warnings for the logger |
| 470033 | Number of warnings forwarded to the logger |

| | |
|---|---|
| 470034 | Max. possible number of error history entries |
| 470035 | Number of entries in the error history |
| 470040 | Error numbers |
| 470041 | Time of the error in ms |
| 470042 | Instance, at which the error occurred |
| 470043 | Number of error parameters |
| 470044 ... 470048 | Error parameters 1 through 5 |
| 470049 | Number of characters of the error message |
| 470050 ... 470157 | Text of the error message |

**First instance**

| | |
|---|---|
| 471010 | Status<br>Bit 0 = 1: Error<br>Bit 1 = 1: Alarms<br>Bit 2 = 1: An instance has been initialized<br>Bit 3 = 1: Execution in process |
| 471011 | Command<br>0: There are no commands |

## JetIPScan

**Global status information**

| | |
|---|---|
| 520000 | Summary of status messages |
| 520010 | State of execution - Corresponds to the feedback value *State*. |
| 520011 | Number of cycles - Corresponds to the feedback value *Count*. |
| 520012 | Number of changes - Corresponds to the feedback value *Changed*. |
| 520013 | Result of the function - Corresponds to the feedback value *Result*. |

**Warnings and errors**

| | |
|---|---|
| 521000 ... 521006 | All 3 responses are dissimilar |
| 521010 ... 521016 | Reply no. 1 is not the same as replies 2 and 3. |
| 521020 ... 521026 | Reply no. 2 is not the same as replies 1 and 3. |
| 521030 ... 521036 | Reply no. 3 is not the same as replies 1 and 2. |
| 521100 ... 521106 | Wrong CtrlID or CtrlIDopt |
| 521200 ... 521206 | The node has not called |
| 521300 ... 521306 | Multiple call |
| 521400 ... 521406 | The IP settings could not be changed |

**Configuration**

| | |
|---|---|
| 522000 | GNN |
| 522010 ... 522015 | Set configuration |
| 522110 ... 522123 | Actual configuration 1 |
| 522210 ... 522223 | Actual configuration 2 |
| 522310 ... 522323 | Actual configuration 3 |

## Application registers

| | |
|---|---|
| 1000000 ... 1001999 | JC-340: 32-bit integer or floating point number (non-volatile) |
| 1000000 ... 1019999 | JC-340: 32-bit integer or floating point number (non-volatile); with option -SD |
| 1000000 ... 1029999 | JC-350: 32-bit integer or floating point number (non-volatile) |
| 1000000 ... 1059999 | JC-360: 32-bit integer or floating point number (non-volatile) |
| 1000000 ... 1119999 | JC-360: 32-bit integer or floating point number (non-volatile); with option -R |
| 1000000 ... 1059999 | JC-365: 32-bit integer or floating point number (non-volatile) |
| 1000000 ... 1119999 | JC-365: 32-bit integer or floating point number (non-volatile); with option -R |

## JX3 system bus registers

| | |
|---|---|
| 100000000 | Bus status<br>Bit 15 = 1: Data exchange takes place via JX3 system bus. |
| 100002000 | Hardware revision of the JX3 system bus |
| 100002008 | Error register (bit-coded)<br>Bit 3 = 1: Error at module access<br>Bit 16 = 1: Fatal irrecoverable error has occurred. Data interchange has been aborted. |
| 100002011 | I/O module number where error has occurred |
| 100002013 | Number of detected JX3 modules |
| 100002015 | Index to module array |
| 100002016 | Module array |
| 100002023 | Dummy modules |
| 100002034 | Number of retries |
| 100002072 | Version of the JX3 system bus driver |
| 100002111 | Module register number where error has occurred |
| 100002764 | Timeout period for register access [ms] |
| 100003xx0 ... 100003xx9 | Registers on I/O modules<br>(compatibility mode)<br>xx: Module number - 2 (00 ... 15) |
| 100004000 ... 100004367 | Inputs/outputs mapped to registers<br>(see below) |
| 100xx0000 … 100xx9999 | Registers on I/O modules<br>(direct access)<br>xx: Module number (02 ... 17) |

## JX2 system bus registers

| | |
|---|---|
| 200002000 | Version of JX2 system bus driver (IP) |
| 200002008 | Error (bit-coded)<br>Bit 3: I/O or CANopen® module timeout<br>Bit 4: JX2 slave module timeout<br>Bit 9: Error of I/O module periphery<br>Bit 12: Object length has not been set<br>Bit 13: Error during JX2 system bus initialization<br>Bit 14: Timeout of system registers<br>Bit 15: SDO abort |
| 200002011 | I/O module number at timeout |
| 200002012 | JX2 slave module number at timeout |
| 200002013 | Amount of connected I/O modules |
| 200002014 | Amount of connected JX2 slave modules |
| 200002015 | Index to module array |
| 200002016 | Module array |
| 200002023 | Dummy I/O module |
| 200002024 | JX2 slave dummy modules |
| 200002028 | Monitoring interval for I/O modules [10 ms] |
| 200002029 | Baud rate of JX2 system bus |

| | |
|---|---|
| 200002032 | ON delay |
| 200002039 | I/O module where a peripheral fault has occurred (bit-coded) |
| 200002070 | Number of CANopen® modules |
| 200002071 | Actual I/O sum of modules on the JX2 system bus |
| 200002072 | Version of JX2 system bus driver (IP) |
| 200002073 | Timeout for register access to CANopen® modules |
| 200002074 | CANopen® SYNC interval [ms] |
| 200002077 | Enabling JX2 system bus special functions<br>**JC-340 and JC-350:**<br>Bit 2: CAN-Prim in addition to JX2 system bus<br>Bit 3: CAN-Prim only<br>Bit 4: CAN-IDs 0x081 ... 9x09F for CAN-Prim<br>**JC-360(MC) und JC-365(MC):**<br>Bit 3, 2 = 01: CAN-Prim in addition to JX2 system bus<br>Bit 3, 2 = 10: CANopen® interface only (CANopen®-STX-API)<br>Bit 3, 2 = 11: CAN-Prim only<br>Bit 4 = 1: CAN-IDs 0x081 ... 9x09F for CAN-Prim<br>Bit 6 = 1: CANopen® feature in the JX2 system bus driver is deactivated<br>Bit 6 makes sense only if bit 3 has not been set |
| 200002080 | CANopen® module index for JX2 system bus application registers |
| 200002085 | SysBus application registers: Register number (65-89) |
| 200002086 | SysBus application registers: Object ID |
| 200002087 | SysBus application registers: Sub-index |
| 200002088 | SysBus application registers: Length |
| 200002760 | Max. number of I/O update retries |
| 200002761 | Index to array of I/O retry counters |
| 200002762 | Array of I/O retry counters |
| 200002763 | Timeout for I/O update of I/O modules [ms] |
| 200002764 | Timeout for register access to I/O modules [ms] |
| 200002765 | Timeout for register access to JX2 slave modules [ms] |
| 200002821 | Write 1 to set the CAN error counters to 0 |
| 200002824 | Counter for stuff errors |
| 200002825 | Counter for CRC errors |
| 200002826 | Counter for formal errors |
| 200002827 | Counter for acknowledge errors |
| 200002828 | Counter for bit errors |
| 200002995 | Bootloader version of JX2 system bus interface |
| 200003xx0 ... 200003xx9 | Registers on I/O modules<br>xx: I/O module number - 2 (00...22) |
| 200004000 ... 200004367 | Inputs/outputs mapped to registers (see below) |
| 200005x00 ... 200006x99 | I/O registers: CANopen®/JX-SIO<br>x: I/O module number - 70 (0...9) |
| 200007x00 ... 200007x99 | Configuration registers - CANopen®/JX-SIO<br>x: I/O module number - 70 (0...9) |
| 2000xx100 ... 2000xx999 | JX2 slave registers<br>xx: JX2-Slave-Nummer + 10 |

## CAN-Prim registers

| | |
|---|---|
| 200010500 | Status register<br>Bit 1 = 1: CAN message has been received<br>Bit 2 = 0: CAN-ID 11 bits<br>Bit 2 = 1: CAN-ID 29 bits |
| 200010501 | Command register<br>**Direct access**<br>7: Clear the FIFO buffer<br>8: Set CAN-ID to 11 bits<br>9: Set CAN-ID to 29 bits<br>10: Check boxes for received messages<br>**Indirect access**<br>1: Enable the message box<br>2: Disable the message box<br>3: Send a CAN message<br>4: Clear the NEW DAT bit<br>5: Clear the OVERRUN bit<br>6: Clear the transmit error bit<br>7: Clear the FIFO buffer<br>8: Set CAN-ID to 11 bits<br>9: Set CAN-ID to 29 bits<br>10: Check boxes for received messages |
| 200010502 | Message box number (indirect access) |
| 200010503 | FIFO buffer filling level |
| 200010504 | FIFO data |
| 200010506 | Global receive mask |
| 200010507 | Global receive ID |
| 200010509 | CAN-Prim version (IP) |
| | **Indirect access** |
| 200010510 | Message box status register |
| 200010511 | Message box configuration register |
| 200010512 | CAN-ID |
| 200010513 | Number of data bytes |
| 200010514 ... 200010521 | Data bytes 0 through 7 |
| | **Direct access** |
| 200010530 + box number * 20 | Message box status register |
| 200010531 + box number * 20 | Message box configuration register |
| 200010532 + box number * 20 | CAN-ID |
| 200010533 + box number * 20 | Number of data bytes |
| 200010534 ... 200010541 + box number * 20 | Data bytes |
| 200010542 + box number * 20 | CAN-ID mask |
| 200010543 + box number * 20 | Box command register |
| 200010544 + box number * 20 | Received CAN-ID |

## Inputs/outputs

| | |
|---|---|
| 20001 ... 36000 | Virtual I/Os for RemoteScan |
| 10000xx01 ... 10000xx16 | JX3 modules (xx: 02 ... 17) |
| 20000xx01 ... 20000xx16 | JX2 modules (xx: 02 ... 24) |
| 1nnn01xx01 ... 1nnn01xx16 | JX3 modules via JX3-BN-ETH<br>GNN: 000 ... 199<br>xx: 02 ... 24) |

## 32 combined inputs

**JX3 system bus: + 100000000**

**JX2 system bus: + 200000000**

**Network: + 1GNN910000**

| | | | | |
|---|---|---|---|---|
| 4000 | 101..108 | 109..116 | 201..208 | 209..216 |
| 4001 | 109..116 | 201..208 | 209..216 | 301..308 |
| 4002 | 201..208 | 209..216 | 301..308 | 309..316 |
| 4003 | 209..216 | 301..308 | 309..316 | 401..408 |
| 4004 | 301..308 | 309..316 | 401..408 | 409..416 |
| 4005 | 309..316 | 401..408 | 409..416 | 501..508 |
| 4006 | 401..408 | 409..416 | 501..508 | 509..516 |
| 4007 | 409..416 | 501..508 | 509..516 | 601..608 |
| 4008 | 501..508 | 509..516 | 601..608 | 609..616 |
| 4009 | 509..516 | 601..608 | 609..616 | 701..708 |
| 4010 | 601..608 | 609..616 | 701..708 | 709..716 |
| 4011 | 609..616 | 701..708 | 709..716 | 801..808 |
| 4012 | 701..708 | 709..716 | 801..808 | 809..816 |
| 4013 | 709..716 | 801..808 | 809..816 | 901..908 |
| 4014 | 801..808 | 809..816 | 901..908 | 909..916 |
| 4015 | 809..816 | 901..908 | 909..916 | 1001..1008 |
| 4016 | 901..908 | 909..916 | 1001..1008 | 1009..1016 |
| 4017 | 909..916 | 1001..1008 | 1009..1016 | 1101..1108 |
| 4018 | 1001..1008 | 1009..1016 | 1101..1108 | 1109..1116 |
| 4019 | 1009..1016 | 1101..1108 | 1109..1116 | 1201..1208 |
| 4020 | 1101..1108 | 1109..1116 | 1201..1208 | 1209..1216 |
| 4021 | 1109..1116 | 1201..1208 | 1209..1216 | 1301..1308 |
| 4022 | 1201..1208 | 1209..1216 | 1301..1308 | 1309..1316 |
| 4023 | 1209..1216 | 1301..1308 | 1309..1316 | 1401..1408 |
| 4024 | 1301..1308 | 1309..1316 | 1401..1408 | 1409..1416 |
| 4025 | 1309..1316 | 1401..1408 | 1409..1416 | 1501..1508 |
| 4026 | 1401..1408 | 1409..1416 | 1501..1508 | 1509..1516 |
| 4027 | 1409..1416 | 1501..1508 | 1509..1516 | 1601..1608 |
| 4028 | 1501..1508 | 1509..1516 | 1601..1608 | 1609..1616 |
| 4029 | 1509..1516 | 1601..1608 | 1609..1616 | 1701..1708 |
| 4030 | 1601..1608 | 1609..1616 | 1701..1708 | 1709..1716 |
| 4031 | 1609..1616 | 1701..1708 | 1709..1716 | 1801..1808 |
| 4032 | 1701..1708 | 1709..1716 | 1801..1808 | 1809..1816 |
| 4033 | 1709..1716 | 1801..1808 | 1809..1816 | 1901..1908 |
| 4034 | 1801..1808 | 1809..1816 | 1901..1908 | 1909..1916 |
| 4035 | 1809..1816 | 1901..1908 | 1909..1916 | 2001..2008 |
| 4036 | 1901..1908 | 1909..1916 | 2001..2008 | 2009..2016 |
| 4037 | 1909..1916 | 2001..2008 | 2009..2016 | 2101..2108 |
| 4038 | 2001..2008 | 2009..2016 | 2101..2108 | 2109..2116 |
| 4039 | 2009..2016 | 2101..2108 | 2109..2116 | 2201..2208 |
| 4040 | 2101..2108 | 2109..2116 | 2201..2208 | 2209..2216 |
| 4041 | 2109..2116 | 2201..2208 | 2209..2216 | 2301..2308 |
| 4042 | 2201..2208 | 2209..2216 | 2301..2308 | 2309..2316 |
| 4043 | 2209..2216 | 2301..2308 | 2309..2316 | 2401..2408 |
| 4044 | 2301..2308 | 2309..2316 | 2401..2408 | 2409..2416 |

## 16 combined inputs

**JX3-Systembus: + 100000000**

**JX2-Systembus: + 200000000**

**Netzwerk: + 1GNN910000**

| | | | |
|---|---|---|---|
| 4060 | 101..108 | 109..116 | |
| 4061 | 109..116 | 201..208 | |
| 4062 | 201..208 | 209..216 | |
| 4063 | 209..216 | 301..308 | |
| 4064 | 301..308 | 309..316 | |
| 4065 | 309..316 | 401..408 | |
| 4066 | 401..408 | 409..416 | |
| 4067 | 409..416 | 501..508 | |
| 4068 | 501..508 | 509..516 | |
| 4069 | 509..516 | 601..608 | |
| 4070 | 601..608 | 609..616 | |
| 4071 | 609..616 | 701..708 | |
| 4072 | 701..708 | 709..716 | |
| 4073 | 709..716 | 801..808 | |
| 4074 | 801..808 | 809..816 | |
| 4075 | 809..816 | 901..908 | |
| 4076 | 901..908 | 909..916 | |
| 4077 | 909..916 | 1001..1008 | |
| 4078 | 1001..1008 | 1009..1016 | |
| 4079 | 1009..1016 | 1101..1108 | |
| 4080 | 1101..1108 | 1109..1116 | |
| 4081 | 1109..1116 | 1201..1208 | |
| 4082 | 1201..1208 | 1209..1216 | |
| 4083 | 1209..1216 | 1301..1308 | |
| 4084 | 1301..1308 | 1309..1316 | |
| 4085 | 1309..1316 | 1401..1408 | |
| 4086 | 1401..1408 | 1409..1416 | |
| 4087 | 1409..1416 | 1501..1508 | |
| 4088 | 1501..1508 | 1509..1516 | |
| 4089 | 1509..1516 | 1601..1608 | |
| 4090 | 1601..1608 | 1609..1616 | |
| 4091 | 1609..1616 | 1701..1708 | |
| 4092 | 1701..1708 | 1709..1716 | |
| 4093 | 1709..1716 | 1801..1808 | |
| 4094 | 1801..1808 | 1809..1816 | |
| 4095 | 1809..1816 | 1901..1908 | |
| 4096 | 1901..1908 | 1909..1916 | |
| 4097 | 1909..1916 | 2001..2008 | |
| 4098 | 2001..2008 | 2009..2016 | |
| 4099 | 2009..2016 | 2101..2108 | |
| 4100 | 2101..2108 | 2109..2116 | |
| 4101 | 2109..2116 | 2201..2208 | |
| 4102 | 2201..2208 | 2209..2216 | |
| 4103 | 2209..2216 | 2301..2308 | |
| 4104 | 2301..2308 | 2309..2316 | |
| 4105 | 2309..2316 | 2401..2408 | |
| 4106 | 2401..2408 | 2409..2416 | |

## 8 combined inputs

**JX3-Systembus: + 100000000**

**JX2-Systembus: + 200000000**

**Netzwerk: + 1GNN910000**

| | |
|---|---|
| 4120 | 101..108 |
| 4121 | 109..116 |
| 4122 | 201..208 |
| 4123 | 209..216 |
| 4124 | 301..308 |
| 4125 | 309..316 |
| 4126 | 401..408 |
| 4127 | 409..416 |
| 4128 | 501..508 |
| 4129 | 509..516 |
| 4130 | 601..608 |
| 4131 | 609..616 |
| 4132 | 701..708 |
| 4133 | 709..716 |
| 4134 | 801..808 |
| 4135 | 809..816 |
| 4136 | 901..908 |
| 4137 | 909..916 |
| 4138 | 1001..1008 |
| 4139 | 1009..1016 |
| 4140 | 1101..1108 |
| 4141 | 1109..1116 |
| 4142 | 1201..1208 |
| 4143 | 1209..1216 |
| 4144 | 1301..1308 |
| 4145 | 1309..1316 |
| 4146 | 1401..1408 |
| 4147 | 1409..1416 |
| 4148 | 1501..1508 |
| 4149 | 1509..1516 |
| 4150 | 1601..1608 |
| 4151 | 1609..1616 |
| 4152 | 1701..1708 |
| 4153 | 1709..1716 |
| 4154 | 1801..1808 |
| 4155 | 1809..1816 |
| 4156 | 1901..1908 |
| 4157 | 1909..1916 |
| 4158 | 2001..2008 |
| 4159 | 2009..2016 |
| 4160 | 2101..2108 |
| 4161 | 2109..2116 |
| 4162 | 2201..2208 |
| 4163 | 2209..2216 |
| 4164 | 2301..2308 |
| 4165 | 2309..2316 |
| 4166 | 2401..2408 |
| 4167 | 2409..2416 |

## 32 combined outputs

**JX3-Systembus: + 100000000**

**JX2-Systembus: + 200000000**

**Netzwerk: + 1GNN910000**

| | | | | |
|---|---|---|---|---|
| 4200 | 101..108 | 109..116 | 201..208 | 209..216 |
| 4201 | 109..116 | 201..208 | 209..216 | 301..308 |
| 4202 | 201..208 | 209..216 | 301..308 | 309..316 |
| 4203 | 209..216 | 301..308 | 309..316 | 401..408 |
| 4204 | 301..308 | 309..316 | 401..408 | 409..416 |
| 4205 | 309..316 | 401..408 | 409..416 | 501..508 |
| 4206 | 401..408 | 409..416 | 501..508 | 509..516 |
| 4207 | 409..416 | 501..508 | 509..516 | 601..608 |
| 4208 | 501..508 | 509..516 | 601..608 | 609..616 |
| 4209 | 509..516 | 601..608 | 609..616 | 701..708 |
| 4210 | 601..608 | 609..616 | 701..708 | 709..716 |
| 4211 | 609..616 | 701..708 | 709..716 | 801..808 |
| 4212 | 701..708 | 709..716 | 801..808 | 809..816 |
| 4213 | 709..716 | 801..808 | 809..816 | 901..908 |
| 4214 | 801..808 | 809..816 | 901..908 | 909..916 |
| 4215 | 809..816 | 901..908 | 909..916 | 1001..1008 |
| 4216 | 901..908 | 909..916 | 1001..1008 | 1009..1016 |
| 4217 | 909..916 | 1001..1008 | 1009..1016 | 1101..1108 |
| 4218 | 1001..1008 | 1009..1016 | 1101..1108 | 1109..1116 |
| 4219 | 1009..1016 | 1101..1108 | 1109..1116 | 1201..1208 |
| 4220 | 1101..1108 | 1109..1116 | 1201..1208 | 1209..1216 |
| 4221 | 1109..1116 | 1201..1208 | 1209..1216 | 1301..1308 |
| 4222 | 1201..1208 | 1209..1216 | 1301..1308 | 1309..1316 |
| 4223 | 1209..1216 | 1301..1308 | 1309..1316 | 1401..1408 |
| 4224 | 1301..1308 | 1309..1316 | 1401..1408 | 1409..1416 |
| 4225 | 1309..1316 | 1401..1408 | 1409..1416 | 1501..1508 |
| 4226 | 1401..1408 | 1409..1416 | 1501..1508 | 1509..1516 |
| 4227 | 1409..1416 | 1501..1508 | 1509..1516 | 1601..1608 |
| 4228 | 1501..1508 | 1509..1516 | 1601..1608 | 1609..1616 |
| 4229 | 1509..1516 | 1601..1608 | 1609..1616 | 1701..1708 |
| 4230 | 1601..1608 | 1609..1616 | 1701..1708 | 1709..1716 |
| 4231 | 1609..1616 | 1701..1708 | 1709..1716 | 1801..1808 |
| 4232 | 1701..1708 | 1709..1716 | 1801..1808 | 1809..1816 |
| 4233 | 1709..1716 | 1801..1808 | 1809..1816 | 1901..1908 |
| 4234 | 1801..1808 | 1809..1816 | 1901..1908 | 1909..1916 |
| 4235 | 1809..1816 | 1901..1908 | 1909..1916 | 2001..2008 |
| 4236 | 1901..1908 | 1909..1916 | 2001..2008 | 2009..2016 |
| 4237 | 1909..1916 | 2001..2008 | 2009..2016 | 2101..2108 |
| 4238 | 2001..2008 | 2009..2016 | 2101..2108 | 2109..2116 |
| 4239 | 2009..2016 | 2101..2108 | 2109..2116 | 2201..2208 |
| 4240 | 2101..2108 | 2109..2116 | 2201..2208 | 2209..2216 |
| 4241 | 2109..2116 | 2201..2208 | 2209..2216 | 2301..2308 |
| 4242 | 2201..2208 | 2209..2216 | 2301..2308 | 2309..2316 |
| 4243 | 2209..2216 | 2301..2308 | 2309..2316 | 2401..2408 |
| 4244 | 2301..2308 | 2309..2316 | 2401..2408 | 2409..2416 |

## 16 combined outputs

**JX3-Systembus: + 100000000**

**JX2-Systembus: + 200000000**

**Netzwerk: + 1GNN910000**

| | | |
|---|---|---|
| 4260 | 101..108 | 109..116 |
| 4261 | 109..116 | 201..208 |
| 4262 | 201..208 | 209..216 |
| 4263 | 209..216 | 301..308 |
| 4264 | 301..308 | 309..316 |
| 4265 | 309..316 | 401..408 |
| 4266 | 401..408 | 409..416 |
| 4267 | 409..416 | 501..508 |
| 4268 | 501..508 | 509..516 |
| 4269 | 509..516 | 601..608 |

| | | |
|---|---|---|
| 4270 | 601..608 | 609..616 |
| 4271 | 609..616 | 701..708 |
| 4272 | 701..708 | 709..716 |
| 4273 | 709..716 | 801..808 |
| 4274 | 801..808 | 809..816 |
| 4275 | 809..816 | 901..908 |
| 4276 | 901..908 | 909..916 |
| 4277 | 909..916 | 1001..1008 |
| 4278 | 1001..1008 | 1009..1016 |
| 4279 | 1009..1016 | 1101..1108 |
| 4280 | 1101..1108 | 1109..1116 |
| 4281 | 1109..1116 | 1201..1208 |
| 4282 | 1201..1208 | 1209..1216 |
| 4283 | 1209..1216 | 1301..1308 |
| 4284 | 1301..1308 | 1309..1316 |
| 4285 | 1309..1316 | 1401..1408 |
| 4286 | 1401..1408 | 1409..1416 |
| 4287 | 1409..1416 | 1501..1508 |
| 4288 | 1501..1508 | 1509..1516 |
| 4289 | 1509..1516 | 1601..1608 |
| 4290 | 1601..1608 | 1609..1616 |
| 4291 | 1609..1616 | 1701..1708 |
| 4292 | 1701..1708 | 1709..1716 |
| 4293 | 1709..1716 | 1801..1808 |
| 4294 | 1801..1808 | 1809..1816 |
| 4295 | 1809..1816 | 1901..1908 |
| 4296 | 1901..1908 | 1909..1916 |
| 4297 | 1909..1916 | 2001..2008 |
| 4298 | 2001..2008 | 2009..2016 |
| 4299 | 2009..2016 | 2101..2108 |
| 4300 | 2101..2108 | 2109..2116 |
| 4301 | 2109..2116 | 2201..2208 |
| 4302 | 2201..2208 | 2209..2216 |
| 4303 | 2209..2216 | 2301..2308 |
| 4304 | 2301..2308 | 2309..2316 |
| 4305 | 2309..2316 | 2401..2408 |
| 4306 | 2401..2408 | 2409..2416 |

## 8 combined outputs

**JX3 system bus: + 100000000**

**JX2 system bus: + 200000000**

**Network: + 1GNN910000**

| | |
|---|---|
| 4320 | 101..108 |
| 4321 | 109..116 |
| 4322 | 201..208 |
| 4323 | 209..216 |
| 4324 | 301..308 |
| 4325 | 309..316 |
| 4326 | 401..408 |
| 4327 | 409..416 |
| 4328 | 501..508 |
| 4329 | 509..516 |
| 4330 | 601..608 |
| 4331 | 609..616 |
| 4332 | 701..708 |
| 4333 | 709..716 |
| 4334 | 801..808 |
| 4335 | 809..816 |
| 4336 | 901..908 |
| 4337 | 909..916 |
| 4338 | 1001..1008 |
| 4339 | 1009..1016 |
| 4340 | 1101..1108 |
| 4341 | 1109..1116 |
| 4342 | 1201..1208 |
| 4343 | 1209..1216 |
| 4344 | 1301..1308 |
| 4345 | 1309..1316 |
| 4346 | 1401..1408 |
| 4347 | 1409..1416 |
| 4348 | 1501..1508 |
| 4349 | 1509..1516 |
| 4350 | 1601..1608 |
| 4351 | 1609..1616 |
| 4352 | 1701..1708 |
| 4353 | 1709..1716 |
| 4354 | 1801..1808 |
| 4355 | 1809..1816 |
| 4356 | 1901..1908 |
| 4357 | 1909..1916 |
| 4358 | 2001..2008 |
| 4359 | 2009..2016 |
| 4360 | 2101..2108 |
| 4361 | 2109..2116 |
| 4362 | 2201..2208 |
| 4363 | 2209..2216 |
| 4364 | 2301..2308 |
| 4365 | 2309..2316 |
| 4366 | 2401..2408 |
| 4367 | 2409..2416 |

## Special flags for networks

| | |
|---|---|
| 2075 | Error in networking via JetIP |
| 2080 | Ethernet system bus error in R 200008 |
| 2081 | Ethernet system bus error |

## Special flags - Interface monitoring

| | |
|---|---|
| 2088 | OS flag - JetIP |
| 2089 | User flag - JetIP |
| 2090 | OS flag - SER |
| 2091 | User flag - SER |
| 2098 | OS flag - Debug server |
| 2099 | User flag - Debug server |

## Special flags - HMIs

**does not apply to LCD 27**

| | |
|---|---|
| 2160 | [0] |
| 2161 | [1] |
| 2162 | [2] |
| 2163 | [3] |
| 2164 | [4] |
| 2165 | [5] |
| 2166 | [6] |
| 2167 | [7] |
| 2168 | [8] |

| | |
|---|---|
| 2169 | [9] |
| 2170 | [SHIFT]+[0] |
| 2171 | [SHIFT]+[1] |
| 2172 | [SHIFT]+[2] |
| 2173 | [SHIFT]+[3] |
| 2174 | [SHIFT]+[4] |
| 2175 | [SHIFT]+[5] |
| 2176 | [SHIFT]+[6] |
| 2177 | [SHIFT]+[7] |
| 2178 | [SHIFT]+[8] |
| 2179 | [SHIFT]+[9] |
| 2181 | [SHIFT]+[F1] |
| 2182 | [SHIFT]+[F2] |
| 2183 | [SHIFT]+[F3] |
| 2184 | [SHIFT]+[F4] |
| 2185 | [SHIFT]+[F5] |
| 2186 | [SHIFT]+[F6] |
| 2187 | [SHIFT]+[F7] |
| 2188 | [SHIFT]+[F8] |
| 2189 | [SHIFT]+[F9] |
| 2190 | [SHIFT]+[F10] |
| 2191 | [SHIFT]+[F11] |
| 2192 | [SHIFT]+[F12] |
| 2193 | [SHIFT]+[←] |
| 2194 | [SHIFT]+[→] |
| 2195 | [SHIFT]+[R] |
| 2196 | [SHIFT]+[I/O] |
| 2197 | [SHIFT]+[=] |
| 2198 | [SHIFT]+[C] |
| 2199 | [SHIFT]+[ENTER] ([↵]) |
| 2200 | [SHIFT] |
| 2201 | [F1] |
| 2202 | [F2] |
| 2203 | [F3] |
| 2204 | [F4] |
| 2205 | [F5] |
| 2206 | [F6] |
| 2207 | [F7] |
| 2208 | [F8] |
| 2209 | [F9] |
| 2210 | [F10] |
| 2211 | [F11] |
| 2212 | [F12] |
| 2213 | [→] |
| 2214 | [←] |
| 2215 | [R] |
| 2216 | [I/O] |
| 2217 | [=] |
| 2218 | [C] |
| 2219 | [ENTER] ([↵]) |
| 2220 | [-] |
| 2221 | [SHIFT]+[-] |
| 2222 | [.] |
| 2223 | [SHIFT]+[.] |
| 2224 | LED of [F1] |
| 2225 | LED of [F2] |
| 2226 | LED of [F3] |
| 2227 | LED of [F4] |
| 2228 | LED of [F5] |
| 2229 | LED of [F6] |
| 2230 | LED of [F7] |
| 2231 | LED of [F8] |
| 2232 | LED of [F9] |
| 2233 | LED of [F10] |
| 2234 | LED of [F11] |
| 2235 | LED of [F12] |

## Special flags for HMI LCD 27

| | |
|---|---|
| 2209 | [↑] |
| 2210 | [↓] |
| 2211 | [C] |
| 2212 | [↵] |

## Special flags for HMI NUM 25

| | |
|---|---|
| 2186 | [SHIFT]+[S1] |
| 2187 | [SHIFT]+[S2] |
| 2188 | [SHIFT]+[S3] |
| 2189 | [SHIFT]+[S4] |
| 2190 | [SHIFT]+[S5] |
| 2206 | [S1] |
| 2207 | [S2] |
| 2208 | [S3] |
| 2209 | [S4] |
| 2210 | [S5] |

## 32 combined flags

| | |
|---|---|
| 203100 | 0 ... 31 |
| 203101 | 32 ... 63 |
| 203102 | 64 ... 95 |
| 203103 | 96 ... 127 |
| 203104 | 128 ... 159 |
| 203105 | 160 ... 191 |
| 203106 | 192 ... 223 |
| 203107 | 224 ... 255 |

## 16 combined flags

| | |
|---|---|
| 203108 | 0 ... 15 |
| 203109 | 16 ... 31 |
| 203110 | 32 ... 47 |
| 203111 | 48 ... 63 |
| 203112 | 64 ... 79 |
| 203113 | 80 ... 95 |
| 203114 | 96 ... 111 |
| 203115 | 112 ... 127 |
| 203116 | 128 ... 143 |
| 203117 | 144 ... 159 |
| 203118 | 160 ... 175 |

| | |
|---|---|
| 203119 | 176 ... 191 |
| 203120 | 192 ... 207 |
| 203121 | 208 ... 223 |
| 203122 | 224 ... 239 |
| 203123 | 240 ... 255 |

### 32 combined special flags

| | |
|---|---|
| 203124 | 2048 ... 2079 |
| 203125 | 2080 ... 2111 |
| 203126 | 2112 ... 2143 |
| 203127 | 2144 ... 2175 |
| 203128 | 2176 ... 2207 |
| 203129 | 2208 ... 2239 |
| 203130 | 2240 ... 2271 |
| 203131 | 2272 ... 2303 |

### 16 combined special flags

| | |
|---|---|
| 203132 | 2048 ... 2063 |
| 203133 | 2064 ... 2079 |
| 203134 | 2080 ... 2095 |
| 203135 | 2096 ... 2111 |
| 203136 | 2112 ... 2127 |
| 203137 | 2128 ... 2143 |
| 203138 | 2144 ... 2159 |
| 203139 | 2160 ... 2175 |
| 203140 | 2176 ... 2191 |
| 203141 | 2192 ... 2207 |
| 203142 | 2208 ... 2223 |
| 203143 | 2224 ... 2239 |
| 203144 | 2240 ... 2255 |
| 203145 | 2256 ... 2271 |
| 203146 | 2272 ... 2287 |
| 203147 | 2288 ... 2303 |

### Overlaid application registers/flags

| | |
|---|---|
| 1000000 | 256 ... 287 |
| 1000001 | 288 ... 319 |
| 1000002 | 320 ... 351 |
| 1000003 | 352 ... 383 |
| 1000004 | 384 ... 415 |
| 1000005 | 416 ... 447 |
| 1000006 | 448 ... 479 |
| 1000007 | 480 ... 511 |
| 1000008 | 512 ... 543 |
| 1000009 | 544 ... 575 |
| 1000010 | 576 ... 607 |
| 1000011 | 608 ... 639 |
| 1000012 | 640 ... 671 |
| 1000013 | 672 ... 703 |
| 1000014 | 704 ... 735 |
| 1000015 | 736 ... 767 |
| 1000016 | 768 ... 799 |
| 1000017 | 800 ... 831 |
| 1000018 | 832 ... 863 |
| 1000019 | 864 ... 895 |
| 1000020 | 896 ... 927 |
| 1000021 | 928 ... 959 |
| 1000022 | 960 ... 991 |
| 1000023 | 992 ... 1023 |
| 1000024 | 1024 ... 1055 |
| 1000025 | 1056 ... 1087 |
| 1000026 | 1088 ... 1119 |
| 1000027 | 1120 ... 1151 |
| 1000028 | 1152 ... 1183 |
| 1000029 | 1184 ... 1215 |
| 1000030 | 1216 ... 1247 |
| 1000031 | 1248 ... 1279 |
| 1000032 | 1280 ... 1311 |
| 1000033 | 1312 ... 1343 |
| 1000034 | 1344 ... 1375 |
| 1000035 | 1376 ... 1407 |
| 1000036 | 1408 ... 1439 |
| 1000037 | 1440 ... 1471 |
| 1000038 | 1472 ... 1503 |
| 1000039 | 1504 ... 1535 |
| 1000040 | 1536 ... 1567 |
| 1000041 | 1568 ... 1599 |
| 1000042 | 1600 ... 1631 |
| 1000043 | 1632 ... 1663 |
| 1000044 | 1664 ... 1695 |
| 1000045 | 1696 ... 1727 |
| 1000046 | 1728 ... 1759 |
| 1000047 | 1760 ... 1791 |
| 1000048 | 1792 ... 1823 |
| 1000049 | 1824 ... 1855 |
| 1000050 | 1856 ... 1887 |
| 1000051 | 1888 ... 1919 |
| 1000052 | 1920 ... 1951 |
| 1000053 | 1952 ... 1983 |
| 1000054 | 1984 ... 2015 |
| 1000055 | 2016 ... 2047 |

## System function

For reasons of compatibility, the system functions are listed below. In JetSym STX, use the corresponding JetSym STX functions instead of system functions.

| | |
|---|---|
| 4 | Conversion from BCD to HEX |
| 5 | Conversion from HEX to BCD |
| 20 | Square root |
| 21 | Sine |
| 22 | Cosine |
| 23 | Tangent |
| 24 | Arc sine |
| 25 | Arc cosine |
| 26 | Arc tangent |
| 27 | Exponential function |
| 28 | Natural logarithm |
| 29 | Absolute value |
| 30 | Separation of digits before and after the decimal point |
| 50 | Sorting register values |
| 60 | CRC generation for Modbus RTU |
| 61 | CRC check for Modbus RTU |
| 65/67 | Reading register block via Modbus/TCP |
| 66/68 | Writing register block via Modbus/TCP |
| 80/85 | Initializing RemoteScan |
| 81 | Starting RemoteScan |
| 82 | Stopping RemoteScan |
| 90 | Writing a data file |
| 91 | Appending a data file |
| 92 | Reading a data file |
| 96 | Deleting a data file |
| 110 | Sending an e-mail |
| 150 | Configuring NetCopyList |
| 151 | Deleting NetCopyList |
| 152 | Sending NetCopyList |

## JetSym STX functions

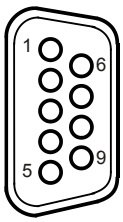
| System function | Corresponding JetSym STX function |
|---|---|
| 4 | Function Bcd2Hex(Bcd: Int): Int; |
| 5 | Function Hex2Bcd(Hex: Int): Int; |
| 50 | Function QSort(DataPtr: Int, ElementCnt: Int, ElementSize: Int, SortOffset: Int, SortType: STXBASETYPE, SortMode: QSORTMODE): Int; |
| 60 | Function ModbusCRCgen(FramePtr: Int, Length: Int): Int; |
| 61 | Function ModbusCRCcheck(FramePtr: Int, Length: Int): Int; |
| 65/67 | Function ModbusReadReg(Const Ref MbParam: MODBUS_PARAM): Int; |
| 66/68 | Function ModbusWriteReg(Const Ref MbParam: MODBUS_PARAM): Int; |
| 80/85 | Function RemoteScanConfig(Protocol: RSCAN_PROTOCOL, Elements: Int, Const Ref Configuration: RSCAN_DSCR): Int; |
| 81 | Function RemoteScanStart(Protocol: Int): Int; |
| 82 | Function RemoteScanStop(Protocol: Int): Int; |
| 90/91 | Function FileDAWrite(Const Ref FileName: String, Const Ref Mode: String, VarType: DAWRITE_TYPE, First: Int, Last: Int): Int; |
| 92 | Function FileDARead(Const Ref FileName: String): Int; |
| 110 | Function EmailSend(Const Ref FileName: String): Int; |
| 150 | Function NetCopyListConfig(IPAddr: Int, IPPort: Int, Const Ref List: TNetCopyLinstL): Int; |
| 151 | Function NetCopyListSend(Handle: Int): Int; |
| 152 | Function NetCopyListDelete(Handle: Int): Int; |

## Assignment of MiniDin port X11




| Pin | Signal | Description |
|---|---|---|
| 1 | RDA | RS-422; receive data inverted |
| 2 | GND | Reference potential |
| 3 | RDB | RS-422; receive data not inverted |
| 4 | RxD | RS-232; receive data |
| 5 | SDB | RS-422; transmit data not inverted<br>RS-485; transmit/receive data not inverted |
| 6 | DC24V | HMI supply voltage |
| 7 | SDA | RS-422; transmit data inverted<br>RS-485; transmit/receive data inverted |
| 8 | TxD | RS-232; transmit data |

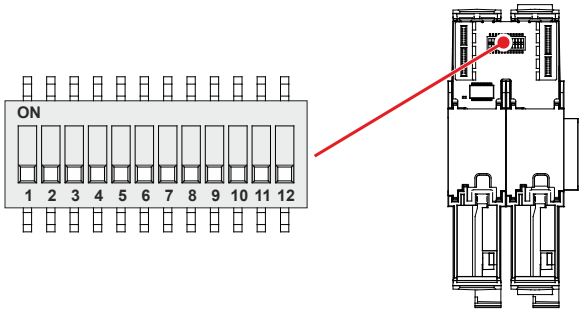
## Pin assignment of female Sub-D connector X19




| Pin | Signal | Description |
|---|---|---|
| 1 | CMODE0 | Commissioning |
| 2 | CAN-L | Data signal for CAN bus 1 |
| 3 | GND | Reference potential |
| 4 | CMODE1 | Commissioning |
| 5 | Unused | |
| 6 | CAN-L_2 | For JC-365(MC):<br>Data signal for CAN bus 2 |
| 7 | CAN-H | Data signal for CAN bus 1 |
| 8 | CAN-H_2 | For JC-365(MC):<br>Data signal for CAN bus 2 |
| 9 | Unused | |

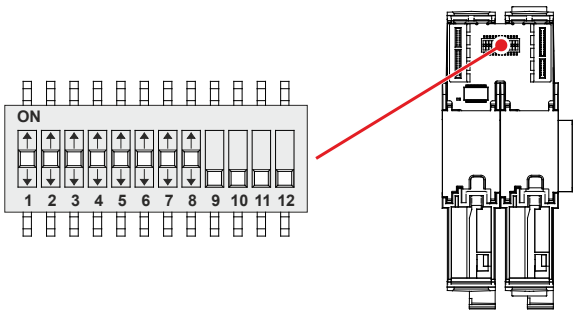
## Setting the default IP address

To set the module to its default IP address 192.168.10.15, move the DIP switch sliders to the positions shown below:




## Setting the IP address via config.ini file and DIP switch

The following DIP switch settings cause the controller to read out the IP address from the file **config.ini** and the DIP switches:




Set the upper three bytes of the IP address in the **config.ini** file, the fourth byte via DIP switch sliders 1 through 8. For further information refer to the user manual.

# Appendix

**Introduction**

This appendix contains electrical and mechanical data, as well as operating data.

**Contents**


| Topic | Page |
|---|---|
| Technical specifications | 638 |
| Index | 646 |

# A: Technical specifications

**Introduction**

This chapter contains information on electrical and mechanical data, as well as on operating data of the JC-360.

**Contents**


| Topic | Page |
|---|---|
| JC-360: Technical data | 639 |
| Physical dimensions | 641 |
| Operating parameters - Environment and mechanics | 642 |
| Operating parameters: Enclosure | 643 |
| DC power supply inputs and outputs | 644 |
| Shielded data and I/O lines | 645 |

# JC-360: Technical data

**Electrical data - Power supply**

| Parameter | Description |
|---|---|
| Rated voltage | DC 24 V |
| Permissible voltage range | -15 % ... +20 % |
| Input current without HMI | 1.0 A max. |
| Input current with HMI | 1.5 A max. |
| Power consumption without HMI | 24 W max. |
| Power consumption with HMI | 36 W max. |

**Technical specifications - JX3 system bus**

The JC-360 supplies the JX3 system bus with logic and additional voltage. The connected JX3 modules are supplied by these two types of voltage.

| Parameter | Description |
|---|---|
| Logic voltage of the JX3 system bus | DC +5 V (-15 % ... +10 %) |
| Additional voltage of JX3 system bus | DC +24 V (-15 % ... +20 %) |

**Data of connected JX3 modules**

The following table shows the maximum current and power consumption of JX3 modules connected to the JC-360.

| Parameter | Description |
|---|---|
| Current consumption absorbed from the logic voltage of the JX3 system bus | $I_{5\,V}$ = 1,200 mA max. |
| Power consumption absorbed from the logic voltage of the JX3 system bus | 6 W max. |
| Current consumption absorbed from the additional voltage of the JX3 system bus | $I_{24\,V}$ = 750 mA max. |
| Power consumption absorbed from the additional voltage of the JX3 system bus | 18 W max. |
| Total power consumption of connected JX3 modules supplied by power supply voltage from the JC-360 | $24\mathrm{V} \cdot I_{24\mathrm{V}} + \frac{5\mathrm{V} \cdot I_{5\mathrm{V}}}{0.85} \leq 18\mathrm{W}$ |

**Memory configurations**

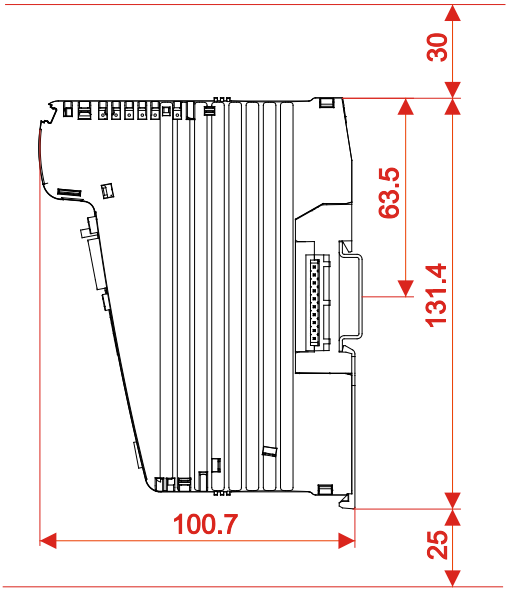
| Parameter | Description |
|---|---|
| Number of non-volatile registers | 60,000<br>Option -R: 120,000 |
| Non-volatile memory for variables | 240,000 bytes<br>Option -R: 480,000 bytes |
| Volatile memory for variables | 4 MB |
| Flash disk | 4 MB |

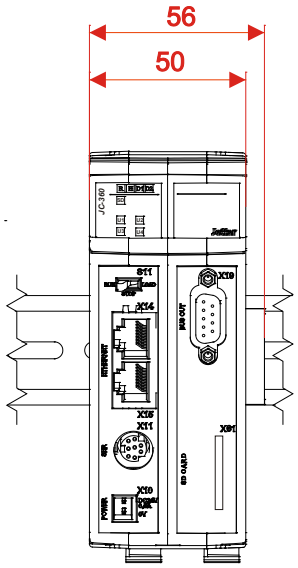
**Technical data - Real-time clock**

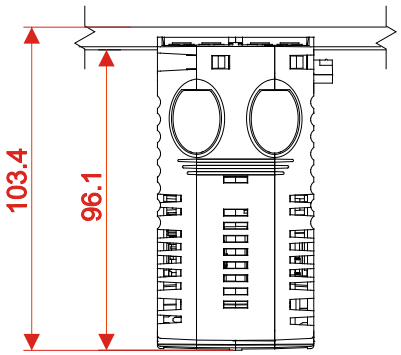
| Parameter | Description |
|---|---|
| Power reserve,<br>if the controller has been running for at least 1 hour. | Minimum: 1 week<br><br>Typical: 2 weeks |
| Deviation | Maximum: 1 min per month |

# Physical dimensions

## Physical dimensions




**Minimum clearances**

At mounting the JC-360, keep the minimum clearance above and below the controller. This way, there must be enough room to press the latches of the JX3 backplane module when replacing modules.

- Minimum clearance, above: 30 mm
- Minimum clearance, below: 25 mm

**Module width**

The width of the JC-360 is 56 mm. When the JC-360 is attached to a JX3 station, its width increases by 50 mm.

**Mounting orientation**

The orientation of the JC-360 is vertical.

# Operating parameters - Environment and mechanics

## Environment

| Parameter | Value | Standard |
|---|---|---|
| Operating temperature range | 5 ... +55 °C | |
| Storage temperature range | -40 ... +70 °C | DIN EN 61131-2<br>DIN EN 60068-2-1<br>DIN EN 60068-2-2 |
| Air humidity | 10 ... 95 %,<br>Non-condensing | DIN EN 61131-2 |
| Pollution degree | 2 | DIN EN 61131-2 |
| Korrosion/<br>Chemical resistance | No special protection against corrosion Ambient air must be free from higher concentrations of acids, alkaline solutions, corrosive agents, salts, metal vapors, or other corrosive or electroconductive contaminants | |
| Maximum operating altitude | 3,000 m above sea level | DIN EN 61131-2 |

## Mechanical parameters

| Parameter | Value | Standard |
|---|---|---|
| Free falls withstanding test | For weight < 10 kg:<br>Height of fall (units within packing): 1 m<br>Product packaging: 0.3 m | DIN EN 61131-2<br>DIN EN 60068-2-31 |
| Vibration resistance | 5...9 Hz: Amplitude 3.5 mm<br>9 ... 150 Hz:<br>1 g acceleration:<br>1 octave/minute,<br>10 sinusoidal frequency sweeps, all three spatial axes | DIN EN 61131-2<br>DIN EN 60068-2-6 |
| Shock resistance | 15 g occasionally, 11 ms, sinusoidal half-wave, 3 shocks in the directions of all three spatial axes | DIN EN 61131-2<br>DIN EN 60068-2-27 |
| Degree of protection | IP20 | DIN EN 60529 |
| Mounting orientation | Vertically snapped on DIN rail | |

# Operating parameters: Enclosure

### Electrical safety

| Parameter | Value | Standard |
|---|---|---|
| Protection class | III | DIN EN 61131-2 |
| Dielectric test voltage | Functional ground is connected to chassis ground internally. | DIN EN 61131-2 |
| Protective connection | 0 | DIN EN 61131-2 |
| Overvoltage category | II | DIN EN 61131-2 |

### EMC - Emitted interference

| Parameter | Value | Standard |
|---|---|---|
| Enclosure | Frequency band 30 ... 230 MHz, limit 30 dB (µV/m) in 10 m<br>Frequency band 230 ... 1,000 MHz, limit 37 dB (µV/m) at 10 m distance<br>(Class B) | DIN EN 61000-6-3<br>DIN EN 61131-2<br>DIN EN 55011 |

### EMC - Interference immunity

| Parameter | Value | Standard |
|---|---|---|
| Magnetic field with mains frequency | 50 Hz<br>30 A/m | DIN EN 61131-2<br>DIN EN 61000-6-2<br>DIN EN 61000-4-8 |
| RF field, amplitude-modulated | Frequency band 80 MHz ... 1 GHz<br>Test field strength: 10 V/m<br>AM 80 % mit 1 kHz<br>Criterion A | DIN EN 61131-2<br>DIN EN 61000-6-2<br>DIN EN 61000-4-3 |
| ESD | Discharge through air:<br>Test peak voltage 8 kV<br>Contact discharge:<br>Test peak voltage 4 kV<br>Criterion A | DIN EN 61131-2<br>DIN EN 61000-6-2<br>DIN EN 61000-4-2 |

# DC power supply inputs and outputs

**EMC - Immunity to interference**

| Parameter | Value | Standard |
|---|---|---|
| RF, asymmetric | Frequency band 0.15 ... 80 MHz<br>Test voltage 10 V<br>AM 80 % with 1 kHz<br>Source impedance 150 Ω<br>Criterion A | DIN EN 61131-2<br>DIN EN 61000-6-2<br>DIN EN 61000-4-6 |
| Bursts | Test voltage 2 kV<br>tr/tn 5/50 ns<br>Repetition rate 5 kHz<br>Criterion A | DIN EN 61131-2<br>DIN EN 61000-6-2<br>DIN EN 61000-4-4 |
| Surge voltages<br>asymmetric (line to earth),<br>symmetrical (line to earth) | tr/th 1.2/50 µs<br>Common-mode interference voltage 1 kV<br>Series-mode interference voltage 0.5 kV | DIN EN 61131-2<br>DIN EN 61000-6-2<br>DIN EN 61000-4-5 |

## Shielded data and I/O lines

**EMC - Immunity to interference**

| Parameter | Value | Standard |
|---|---|---|
| Asymmetric RF, amplitude-modulated | Frequency band 0.15 ... 80 MHz<br>Test voltage 10 V<br>AM 80 % with 1 kHz<br>Source impedance 150 Ω<br>Criterion A | DIN EN 61131-2<br>DIN EN 61000-6-2<br>DIN EN 61000-4-6 |
| Bursts | Test voltage 1 kV<br>tr/tn 5/50 ns<br>Repetition rate 5 kHz<br>Criterion A | DIN EN 61131-2<br>DIN EN 61000-6-2<br>DIN EN 61000-4-4 |
| Voltage surges, asymmetric (line to earth) | tr/th 1.2/50 µs<br>Common-mode interference voltage 1 kV | DIN EN 61131-2<br>DIN EN 61000-6-2<br>DIN EN 61000-4-5 |

# B: Index


## A

Application program
storing to an SD card • 619
Default path • 618
Loading • 620
Factory settings • 72
Automatic copying of controller data • 580
Example of a command file • 597

## C

CANopen® • 521
Changing the IP address • 79
During runtime • 86
Non-volatile, via registers • 84
Operating mode GNN • 87
Setting the default IP address • 80
Via configuration file • 81
Via configuration file and DIP switch • 82
Components of the JC-360 • 21
Configuring a JX3 station • 147
Connecting HMIs • 133
Cable JC-DK-Xm • 137
Cable KAY_0386-xxxx • 139
Cable KAY_0533-0025 • 141
Multi-display mode • 134
Controlling HMIs • 330
Clear display • 346
Connectable alpha-numeric HMIs • 132
Cursor position • 344
Device number • 342
Displaying numerical values • 348
Displaying texts • 340
Entering numerical values • 358
Monitor functions • 382
Querying the keys • 371
Registers - Overview • 335
Controlling serial and printer interfaces • 389
Configuring module numbers • 396
Output of numerical values • 401
Registers - Overview • 394
Outputting texts • 397
Overview - Interfaces • 391

## D

File system • 153
User administration • 157
Properties • 154
Formatting and checking • 171
Sorting data • 465
Disposal • 14
List of documentation • 23

## E

Real-time clock • 313
EDS
EDS file • 31
EDS registers • 35
Inserting real-time controller values • 208
E-mail feature • 441
Configuration • 442
Creating e-mails • 450
Registers - Overview • 461
Sending e-mails • 459
EMC
Notes • 16
Engineering a JX2 station
JX2 system bus • 49
JX2 system bus cable • 51
Line length (in mm) and baud rate • 53
Number of connectable modules • 105
Power supply • 109, 111
Third-party CANopen® modules that can be connected • 112
Engineering a JX3 station
JX3 system bus configurator • 92
Restrictions • 95, 96, 100
Initial commissioning • 143

## F

FTP client • 179
FTP server • 175

## H

Hardware Manager • 272
Open • 273
HTTP server • 205

## I

I/O number
of CANopen® modules on the JX2 system bus • 236
of IP67-I/O modules on the JX2 system bus • 235
of JX2-I/O modules on the JX2 system bus • 234
of JX3 modules connected to JX3-BN-ETH • 237
of JX3 modules within the JX3 station • 232
Inserting control values in a HTML file • 208
Intended conditions of use • 14
Interfaces - Overview • 21
IP configuration
Configuration file • 74
Configuration memory • 73
Configuration registers • 78

## J

Jacks
Connector X19 - JX2 system bus • 49
Jack X11 - Serial interface • 44
Jacks X14 and X15 - Ethernet • 47
JetIPScan
Registers - Description • 279
Configuration • 288

Global status • 281
Register numbers • 280
Warnings and errors • 284
Jetter Ethernet system bus • 240
Cyclic data exchange • 259
Network inputs and outputs • 269
Network registers • 269
Publish/subscribe • 261
Publish/subscribe - Registers • 263
Executing an ARP request • 294
Explicit data exchange • 243
Addressing with variable destination window • 254
Indirect addressing of remote modules • 252
NetBit() • 247
NetCopy() • 245
Registers - Description • 256
Registers located on JX3 modules • 250
Global Node Number • 242
JetSync blockage • 296
System commands • 300
System command register • 297
Locating faults
CRC calculation • 276
Explicit data exchange • 275
Remote node • 278
Subscription • 277
TCP-S - Connection management • 290
JX2 system bus • 408
Coding of modules • 409
Configuring dummy modules • 414
JX2 system bus • 49
JX2 system bus cable • 51
Line length (in mm) and baud rate • 53
Monitoring interval • 415
Power supply • 109, 111
Setting the baud rate • 412
Third-party CANopen® modules that can be connected • 112
JX3 system bus • 431
Coding of modules • 432
Configuring dummy modules • 434
JX3 system bus configurator • 92
Restrictions • 95, 96, 100


## K


Terminals
Terminal X10 - Power supply • 43


## L


LEDs of the controller • 55, 57, 59, 60
Locating faults
CRC calculation • 276
Explicit data exchange • 275
Remote node • 278
Subscription • 277
Warnings and errors • 284


## M


Mechanical installation
Installing the controller JC-360 • 67
Physical dimensions • 26
Memory types • 217
Modbus/TCP • 466
Modbus/TCP client • 473, 475
Modbus/TCP server • 467
Modifications • 14
Monitoring interface activities • 325


## O


Operating parameters
Enclosure • 643
Environment and mechanics • 642
Shielded data and I/O lines • 645
DC power supply inputs and outputs • 644
Operating system update • 605
Order reference - JC-360 • 22


## P


Personnel qualification • 14
Product description
JC-360 • 20


## Q


Quick reference • 621


## R


Register number
of CANopen® modules on the JX2 system bus • 236
of IP67-I/O modules on the JX2 system bus • 235
of JX2 slave modules connected to the JX2 system bus • 233
of JX2-I/O modules on the JX2 system bus • 234
of JX3 modules connected to JX3-BN-ETH • 237
of JX3 modules within the JX3 station • 232
Removing
Removing a controller JC-360 • 70
Replacing the Controller • 68
Repair • 14
Replacing modules • 14
Runtime registers • 322


## S


Safety instructions • 13
Mode selector S11 • 62
SD card slot X61 • 64
System commands • 306
System registers • 297


## T


Technical specifications • 639
Ethernet port • 47
JX2 system bus • 49

Power supply • 43
SD card • 64
Serial interface • 44
Transport • 14
Nameplate • 29

## U

Usage other than intended • 14
User-programmable CAN-Prim interface • 552
Function • 557
Internal process • 558
Registers - Description • 559, 564, 570
Restrictions • 553
Transmitting RTR telegrams • 578
Use • 574
Using CAN ID masks • 577
User-programmable IP interface • 502
Programming the IP interface • 504
Registers - Overview • 517
User-programmable serial interface • 477
Connection • 44
Operating principle • 483
Programming the serial interface • 495
Registers - Overview • 487

## V

Version registers • 37

## Z

Accessories for the JX3 system • 25

Jetter automation

Jetter AG
Graeterstrasse 2
71642 Ludwigsburg | Germany

Phone +49 7141 2550-0
Fax +49 7141 2550-425
info@jetter.de
www.jetter.de

We automate your success.